

Web Services in PHP: da xmlrpc a json e ritorno

Ing. Gaetano Giunta



PHP Day 2006
canale Developer
Bari – 20 maggio 2006

Intro

Programma della presentazione:

- Breve introduzione ai webservices, con focus su XMLRPC, SOAP e JSON
- Creazione di un client xmlrpc
- Creazione di un server xmlrpc
- Estensione del server xmlrpc con supporto per json-rpc
- Estensione del server con supporto per chiamate effettuate direttamente dal browser (AJAX? Mai sentito!)

Dr. Webservice, suppongo?

“a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface that is described in a machine-processible format such as WSDL. Other systems interact with the Web service in a manner prescribed by its interface using messages, which may be enclosed in a SOAP envelope, or follow a REST approach. These messages are typically conveyed using HTTP, and normally comprise XML in conjunction with other Web-related standards”

da wikipedia, maggio 2006

in pratica sta per

“comunicazione tra applicazioni via XML ed HTTP”
(ma né xml né http sono obbligatori...)

XMLRPC (www.xmlrpc.com)

- Il primo nato: spec. del 1999 (implementato nel 1998?)
- Copyright sulla specifica: UserLand SW (Dave Winer)
- “primo utilizzo diffuso di xml su piattaforme diverse”
- Implementazioni esistenti in tutti i linguaggi: javascript, java, C, php, python, perl, ruby, delphi, VB, etc...
- Legato a doppio filo ad HTTP e chiamate sincrone
- Limitate capacità di descrizione delle interfacce esposte
- Non integrato con standard xml successivi (namespaces, xml schema, wsdl)

XMLRPC - sintassi

- Basato su 6 tipi base: stringa, intero, floating point, booleano, base64, data e 2 ricorsivi: array, struttura
- Non fa uso di attributi o namespace
- La sintassi con cui si definiscono i servizi esposti è quella di una chiamata di funzione:

```
risultato = funzione (p1, p2, ..., pn)
```
- È previsto un risultato di tipo “errore” (numero intero + stringa) all'invocazione del metodo
- Xml generato verboso ma “comprensibile”

XMLRPC - esempio

String system.MethodHelp(String methodname)

```
<?xml version="1.0"?>
<methodCall>
<methodName>system.methodHe
lp</methodName>
  <params>
    <param>
      <value>
<string>system.methodHelp</
string>
      </value>
    </param>
  </params>
</methodCall>
```

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <string>Returns
help text if defined for
the method passed,
otherwise returns an empty
string</string>
      </value>
    </param>
  </params>
</methodResponse>
```

SOAP (www.w3.org/TR/soap/)

- Nato per sopperire alle carenze di xmlrpc
- Standard approvato dal W3C (solo la versione 1.2)
- Supportato da Microsoft, IBM, SUN
- Permette di descrivere nel dettaglio le interfacce (WSDL)
- In teoria: permette di svincolare la sintassi di una chiamata dalla sua rappresentazione fisica (e dal protocollo di trasporto)
- In pratica: grossi problemi di interoperabilità sin dall'inizio tra le diverse implementazioni

SOAP - WSDL

- Basato su XML Schema per la definizione dei tipi (in teoria anche Relax NG è utilizzabile, ma non è supportato dalle librerie disponibili)
- Ridondante e verboso per ottenere la necessaria flessibilità...
- ...ma il binding HTTP è l'unico utilizzato da tutti (con rappresentazione document/literal)
- Estremamente complesso da sviluppare “a mano”
- Un “anti-pattern” dei linguaggi di descrizione delle interfacce?

SOAP - esempio

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
  <sendMessages xmlns="SmsServer">
    <outMessageList xmlns="">
      <outputMessage>
        <message>
          <telephoneNumber>1-800-neo</telephoneNumber>
          <messageText>follow the white rabbit</messageText>
        </message>
        <requestConfirmation>false</requestConfirmation>
      </outputMessage>
    </outMessageList>
    <senderID xmlns="">the matrix</senderID>
  </sendMessages>
</soap:Body>
</soap:Envelope>
```

Servizio: invio di
una serie di
SMS

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
  <sendMessagesResponse xmlns="">
    <messageID xmlns="">0</messageID>
  </sendMessagesResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP – il wsdl corrispondente...

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<definitions name="SmsService-interface"
  targetNamespace="SmsServer"
  xmlns:tns="SmsServer"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <documentation>
    Defines the interface used to send / receive SMS messages
  </documentation>
  <types>
    <schema targetNamespace="SmsServer"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:soap="http://schemas.xmlsoap.org/soap/"
      xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:tns="SmsServer">
      <complexType name="smsMessage">
        <sequence>
          <element name="telephoneNumber" type="xsd:string" minOccurs="1" maxOccurs="1"/>
          <element name="messageText" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </sequence>
      </complexType>
      <complexType name="smsOutputMessage">
        <sequence>
          <element name="message" type="tns:smsMessage" minOccurs="1" maxOccurs="1"/>
          <element name="requestConfirmation" type="xsd:boolean" minOccurs="0" maxOccurs="1"/>
        </sequence>
      </complexType>
      <complexType name="smsInputMessage">
        <sequence>
          <element name="message" type="tns:smsMessage" minOccurs="1" maxOccurs="1"/>
          <element name="receptionDate" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
          <element name="messageID" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
        </sequence>
      </complexType>
      <complexType name="outputMessageList">
        <sequence>
          <element name="outputMessage" type="tns:smsOutputMessage" minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
      <complexType name="inputMessageList">
        <sequence>
          <element name="inputMessage" type="tns:smsInputMessage" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
    </schema>
    <element name="sendMessages">
      <complexType>
        <sequence>
          <element name="outMessageList" type="tns:outputMessageList" minOccurs="1" maxOccurs="1"/>
          <element name="senderID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </sequence>
      </complexType>
    </element>
    <element name="sendMessagesResponse">
      <complexType>
        <sequence>
          <element name="messageID" type="xsd:integer" minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
    </element>
  </types>
  <portType name="smsService">
    <operation name="receiveMessages">
      <input message="tns:receiveRequest" name="receiveMessagesRequest"/>
      <output message="tns:receiveResponse" name="receiveMessagesResponse"/>
    </operation>
  </portType>
  <binding name="smsServiceBinding" type="tns:smsService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sendMessages">
      <soap:operation soapAction=""/>
      <input name="sendMessagesRequest">
        <soap:body use="literal"/>
      </input>
      <output name="sendMessagesResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="SmsService">
    <port binding="tns:smsServiceBinding" name="smsServicePortBinding">
      <!--<soap:address location="http://10.1.13.36/seainternet/cms_sms_adodb_server/server.php"/>-->
      <soap:address location="http://10.1.1.230/webservices/seamess/server.php"/>
    </port>
  </service>
</definitions>
```

JSON (www.json.org)

- L'ultimo nato
- Come xmlrpc, si basa su un set limitato di tipi base: numero, stringa, booleano, null, array, oggetto
- Abbandona la notazione xml per una più sintetica e semplice da decodificare
- Rappresenta un sottoinsieme della 'object literal notation', parte della specifica javascript (ECMA 262)
- Descrive esclusivamente un formato di rappresentazione dei dati, non un protocollo di trasporto né un pattern di scambio messaggi

JSON – il partner per AJAX

- Vantaggi evidenti nell'utilizzo per la comunicazione tra browser e server:
 - Formato più conciso di XML: minori dati transitano sulla rete
 - Rapidità nell'esecuzione del parsing dei messaggi in oggetti javascript (non necessita di librerie xml)
 - Semplicità del codice javascript che fa uso dei dati ricevuti (nessuna navigazione di DOM o metodi di accesso similari)

```
var result = eval( json_data );
```

- Attenzione ai grossi problemi di sicurezza insiti nel codice qui sopra!

JSON - esempio

```
{  
  "method": "system.methodHelp",  
  "params": ["system.methodHelp" ],  
  "id": 0  
}
```

```
{  
  "id": 0,  
  "error": null,  
  "result": "Returns help text if defined for the method  
passed, otherwise returns an empty string"  
}
```

Quale protocollo devo usare?

YMMV (in altre parole: dipende...)

- SOAP 'ha vinto' la guerra degli standard ed è la base per molti altri standard: la famiglia WS-*
- Gli altri due sono diffusissimi in internet!
- Si controllano entrambi i lati della comunicazione o uno solo?
- Esistono toolkit stabili e semplici da utilizzare per la piattaforma di sviluppo in uso?
- Privilegiare tool che fanno tutto da soli (protocollo nascosto) o semplici da estendere/modificare?
- Velocità di esecuzione del codice vs. semplicità dello stesso

PHP per i webservices

- Estensione XMLRPC: nel core a partire dal PHP 4
- Estensione SOAP: PHP 5
- Estensione JSON: PHP 6 (forse già 5.2?)
- Oltre alle estensioni native, esistono decine di librerie implementate “in php”:
 - PEAR::xmlrpc, phpxmlrpc, Incutio, Keith Devens, Zend_XmlRpc, ...
 - PEAR::soap, nusoap, ...
 - PEAR::json, phpxmlrpc-extras, JSON-PHP, Zend_json, ...

PHPXMLRPC

- La prima libreria per xmlrpc in php (1999)
- Codice debuggato e stabile*:
 - Solamente 2 files
 - Inclusa in decine di applicazioni PHP
- Supporta numerose features: compressione http, gestione dei character set, cookies, http 1.1, https, autenticazione basic ed ntlm, proxy, ...
- Flessibile: gestione da 'automagica' a 100% manuale della conversione di variabili php in xmlrpc e di funzioni in webservices (e viceversa)
- Include un comodo debugger

* dopo la correzione dei problemi di iniezione di codice del 2005

PHPXMLRPC - extras

- Aggiunge supporto per JSONRPC, un “dialetto” di JSON (<http://json-rpc.org/>)
- Oltre a json, altre funzioni interessanti:
 - Generazione automatica di documentazione HTML delle interfacce
 - Nuove classi per interazioni “ajax” semplificate
 - Generazione del WSDL per i metodi esposti ?
- Codice ancora in fase “beta”

XMLRPC - installazione

- Scaricare il software, da <http://sourceforge.net/projects/phpxmlrpc> sia il pacchetto “phpxmlrpc” che “extras”
- Copiare in una directory che faccia parte del path di inclusione del PHP i seguenti file:
`xmlrpc.inc, xmlrpcs.inc` (libreria base)
`jsonrpc.inc, jsonrpcs.inc, docxmlrpcs.inc, ajaxmlrpc.inc` (estensioni)
- Copiare la cartella `debugger` da qualche parte sotto la web root
- Creare una cartella in cui risiederanno i web services (sotto la web root, ovviamente ;)

XMLRPC - concetti

Classi:

| | |
|----------------------------|---|
| <code>xmlrpcval</code> | un valore xmlrpc (scalare o ricorsivo) |
| <code>xmlrpcmsg</code> | inviato dal client al server |
| <code>xmlrpcresp</code> | inviata dal server al client |
| <code>xmlrpc_client</code> | invia messaggi al server |
| <code>xmlrpc_server</code> | riceve le richieste dai client, le inoltra alle appropriate funzioni utente e gestisce l'invio della risposta |

Esistono inoltre alcune funzioni di utilità generale

Creazione di un client – 1

```
<?php
    include("xmlrpc.inc");

    $par = new xmlrpcval('system.methodHelp', 'string');
    $msg = new xmlrpcmsg('system.methodHelp', array($par));
    $c = new xmlrpc_client("/server.php", "phpxmlrpc.sf.net", 80);
    $r = $c->send($msg);
    if(!$r->faultCode())
    {
        $v = $r->value();
        $result = $v->scalarval(); // non funzionerebbe con array!
        print "Got: " . htmlspecialchars($result);
    }
    else
    {
        print "An error occurred:\n";
        print "  Code: " . htmlspecialchars($r->faultCode())."\n";
        print "  Reason: " . htmlspecialchars($r->faultString());
    }
?>
```

Creazione di un client – 2

La conversione manuale da variabili PHP a oggetti xmlrpcval diventa rapidamente tediosa...

```
$v = new xmlrpcval(array(
    "thearray" => new xmlrpcval(array(
        new xmlrpcval("ABCDEFHIJ"),
        new xmlrpcval(1234, 'int'),
        new xmlrpcval(1, 'boolean'),
        new xmlrpcval(0, 'boolean'),
        new xmlrpcval(true, 'boolean'),
        new xmlrpcval(false, 'boolean')),
        "array"),
    "theint" => new xmlrpcval(23, 'int'),
    "thestring" => new xmlrpcval("foobarwhizz"),
    "thestruct" => new xmlrpcval(array(
        "one" => new xmlrpcval(1, 'int'),
        "two" => new xmlrpcval(2, 'int')),
        "struct")
),
"struct");
```

Creazione di un client - 3

... ma vi si può ovviare facilmente

```
// ...

$par = php_xmlrpc_encode($some_extremely_complex_php_array);

$msg = new xmlrpcmsg('whatever', array($par));
$c = new xmlrpc_client("/server.php", "phpxmlrpc.sf.net", 80);
$r = $c->send($msg);
if(!$r->faultCode())
{
    $result = php_xmlrpc_decode($r->value());
    var_dump($result);
}
else
{
    print "An error occurred:\n";
    print "  Code: " . htmlspecialchars($r->faultCode())."\n";
    print "  Reason: " . htmlspecialchars($r->faultString());
}
}
```

Creazione di un client - 4

Per i veramente pigri: conversione di metodi remoti in funzioni PHP locali

```
$c = new xmlrpc_client("phpxmlrpc.sourceforge.net/server.php");  
$funcname = wrap_xmlrpc_method($c, "system.methodHelp");  
if ($funcname)  
{  
    $methodhelp = $funcname("system.methodHelp");  
    if (is_a($methodhelp, "xmlrpcresp"))  
        print "KO: method invocation failed";  
    else  
        print "Got: " . htmlspecialchars($methodhelp);  
}  
else  
{  
    print "KO: Cannot get method synopsis from server";  
}
```

Creazione di un server - 1

```
<?php
    include('xmlrpc.inc');
    include('xmlrpcs.inc');

    $methods = array(
        'USstatename' => array(
            'function' => 'findstate',
            'signature' => array(array($xmlrpcString,
$xmlrpcInt)),
            'docstring' => 'Converts an integer value to its US
state name' )
        );

    function findstate($msg)
    {
        // funzionalità da implementare...
    }

    $server = new xmlrpc_server($methods);
?>
```

Creazione di un server - 2

Funzione che implementa un metodo: espone un'interfaccia standard

```
function findstate($msg) {
    global $stateNames, $xmlrpcerruser;

    $in = php_xmlrpc_decode($msg);
    $snv = $in[0]; // la dispatch map ci garantisce un int
    if ($snv >= 0 && $snv <= 50)
    {
        $resp = new xmlrpcresp(
            php_xmlrpc_encode($stateNames[$snv-1]));
    } else {
        $resp = new xmlrpcresp(0, $xmlrpcerruser,
            "I don't have a state for the index '" . $snv . "'");
    }
    return $resp;
}
```

Creazione di un server - 3

Per i veramente pigri: conversione di funzioni PHP locali in metodi remoti (solo con PHP 5.0.3 o successivi)

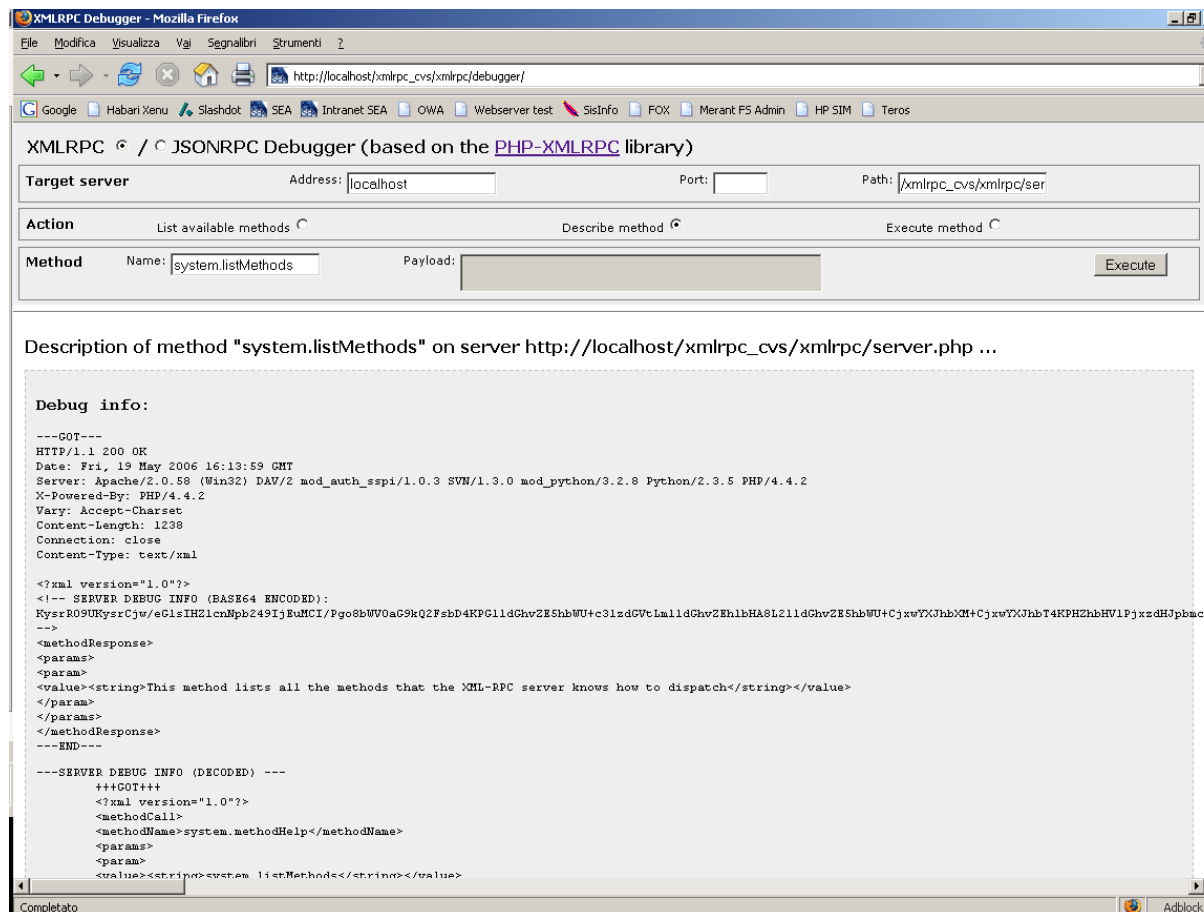
```
/** Converts an integer value to its US state name
 * @param int $snv
 * @return string */
function findstate2($snv) {
    global $stateNames;
    if ($snv >= 0 && $snv <= 50)
        $resp = $stateNames[$snv-1];
    else
        $resp = "I don't have a state for the index ".$snv;
    return $resp;
}

$findtstate_sig = wrap_php_function('findstate2');

if ($findtstate_sig)
    $methods['automagic.findstate'] = $findtstate_sig;
```

Uso del debugger

Problema ricorrente nello sviluppo di webservices (e, in generale, protocolli di comunicazione): l'osservabilità



The screenshot displays the XMLRPC Debugger interface within Mozilla Firefox. The browser's address bar shows the URL `http://localhost/xmlrpc_cvs/xmlrpc/debugger/`. The interface includes a menu bar, a toolbar, and a main content area. The main content area is titled "XMLRPC / JSONRPC Debugger (based on the PHP-XMLRPC library)". It features a "Target server" section with fields for "Address" (localhost), "Port" (empty), and "Path" (`/xmlrpc_cvs/xmlrpc/ser`). Below this is an "Action" section with buttons for "List available methods", "Describe method", and "Execute method". The "Method" section shows the "Name" as `system.listMethods` and a "Payload" field. An "Execute" button is present. The main content area displays the "Description of method 'system.listMethods' on server http://localhost/xmlrpc_cvs/xmlrpc/server.php ...". The description includes "Debug info:" and a detailed XML-RPC response. The response includes headers like `HTTP/1.1 200 OK` and `Date: Fri, 19 May 2006 16:13:59 GMT`. The XML body contains a `<methodResponse>` element with a `<params>` element containing a `<string>` value: `This method lists all the methods that the XML-RPC server knows how to dispatch</string></value>`. The response also includes a `<methodCall>` element with `<methodName>system.methodHelp</methodName>` and a `<params>` element containing a `<string>` value: `system.listMethods</string></value>`.

Documentazione “per umani”

```
<?php
    include('xmlrpc.inc');
    include('xmlrpcs.inc');
    include('docxmlrpcs.inc');

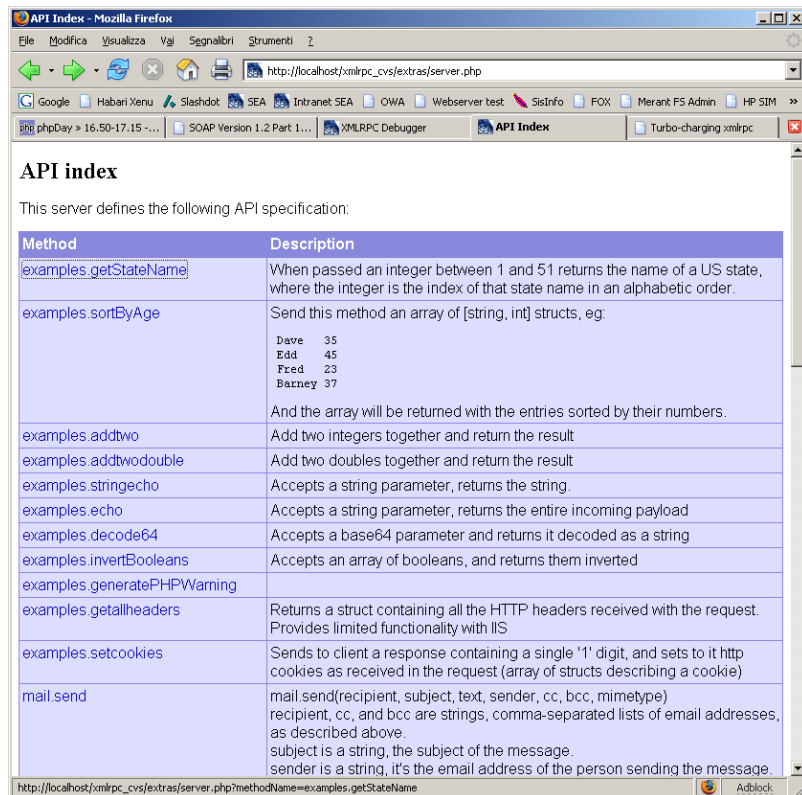
    $methods = array(
        'USstatename' => array(
            'function' => 'findstate',
            'signature' => array(array($xmlrpcString,
$xmlrpcInt)),
            'docstring' => 'Converts an integer value to its US
state name' )
        );

    function findstate($msg)
    {
        // funzionalità da implementare...
    }

    $server = new documenting_xmlrpc_server($methods);
?>
```

Documentazione “per umani”

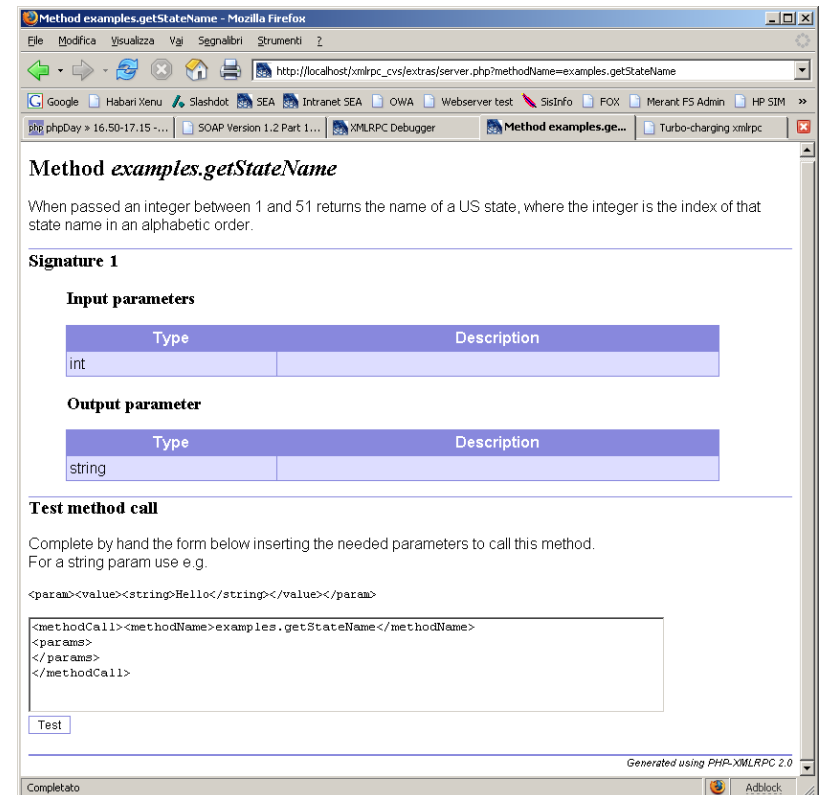
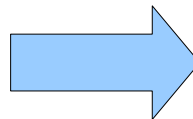
Oltre alla lista dei metodi e alla sintassi di chiamata di ogni metodo, si ottiene un form in cui completare la richiesta (come xml) e testarla immediatamente



API index

This server defines the following API specification:

| Method | Description |
|---|--|
| examples.getStateName | When passed an integer between 1 and 51 returns the name of a US state, where the integer is the index of that state name in an alphabetic order. |
| examples.sortByAge | Send this method an array of {string, int} structs, eg: Dave 35 Edd 45 Fred 23 Barney 37 And the array will be returned with the entries sorted by their numbers. |
| examples.addtwo | Add two integers together and return the result |
| examples.addtwodouble | Add two doubles together and return the result |
| examples.stringecho | Accepts a string parameter, returns the string. |
| examples.echo | Accepts a string parameter, returns the entire incoming payload |
| examples.decode64 | Accepts a base64 parameter and returns it decoded as a string |
| examples.invertBooleans | Accepts an array of booleans, and returns them inverted |
| examples.generatePHPWarning | |
| examples.getAllheaders | Returns a struct containing all the HTTP headers received with the request. Provides limited functionality with IIS |
| examples.setcookies | Sends to client a response containing a single '1' digit, and sets to it http cookies as received in the request (array of structs describing a cookie) |
| mail.send | mail send(recipient, subject, text, sender, cc, bcc, mimetype) recipient, cc, and bcc are strings, comma-separated lists of email addresses, as described above. subject is a string, the subject of the message. sender is a string, it's the email address of the person sending the message. |



Method *examples.getStateName*

When passed an integer between 1 and 51 returns the name of a US state, where the integer is the index of that state name in an alphabetic order.

Signature 1

Input parameters

| Type | Description |
|------|-------------|
| int | |

Output parameter

| Type | Description |
|--------|-------------|
| string | |

Test method call

Complete by hand the form below inserting the needed parameters to call this method. For a string param use e.g.

```
<param><value><string>Hello</string></value></param>
```

```
<methodCall><methodName>examples.getStateName</methodName>
<params>
</params>
</methodCall>
```

Generated using PHP-XMLRPC 2.0

Supporto per json-rpc: le classi

Le classi che forniscono il supporto per json-rpc sono sottoclassi di quelle usate per xml-rpc

| | | |
|----------------------------|----------------|-----------------------------|
| <code>xmlrpcval</code> | <code>=</code> | <code>jsonrpcval</code> |
| <code>xmlrpcmsg</code> | <code>=</code> | <code>jsonrpcmsg</code> |
| <code>xmlrpcresp</code> | <code>=</code> | <code>jsonrpcresp</code> |
| <code>xmlrpc_client</code> | <code>=</code> | <code>jsonrpc_client</code> |
| <code>xmlrpc_server</code> | <code>=</code> | <code>jsonrpc_server</code> |

la “magia” è tutta nella classe server, che riconosce il protocollo usato dal client e serializza le risposte in modo appropriato.

Server multiprotocollo

```
<?php
    include('xmlrpc.inc');
    include('xmlrpcs.inc');
    include('jsonrpc.inc');
    include('jsonrpcs.inc');

    // tutta la dispatch map e le funzioni PHP rimangono
    // invariate...

    $server = new jsonrpc_server($methods);
?>
```

Funziona sempre correttamente se non si utilizzano:

- datetime e base64 per xmlrpc
- null per json

Le funzioni php che implementano metodi non cambiano!

Combinando le potenzialità...

```
if($_SERVER['REQUEST_METHOD'] != 'POST' ||
    $_SERVER['CONTENT_TYPE'] == 'application/x-www-form-
    urlencoded')
{
    include('docxmlrpcs.inc');
    $server = new documenting_xmlrpc_server($methods);
}
else
{
    if strpos($_SERVER['CONTENT_TYPE'], 'text/xml') === 0)
    {
        $server = new xmlrpc_server($methods);
    }
    else
    {
        include('jsonrpc.inc'); include('jsonrpcs.inc');
        $server = new jsonrpc_server($methods);
    }
}
```

AJAX e WS: da PHP a JS

Come si riescono a convertire le funzioni PHP in metodi xmlrpc o jsonrpc, si potrebbero convertire tali metodi in funzioni javascript?

Grazie all'aiuto di Jsolait (<http://jsolait.net/>) si possono invocare i webservices direttamente dal browser

Limitazione di base: si possono invocare webservices solamente sul server originante della pagina web (a meno di orrendi hacks o di creare un proxy)

AJAX e WS: da PHP a JS

```
<script type="text/javascript" src="./jsolait/init.js">
<script type="text/javascript">
    var xmlhttp = importModule("xmlrpc");

    var serviceURL =
        "http://phpxmlrpc.sourceforge.net/server.php";
    var methods = ["USstatename"];
    try{
        var service = new xmlhttp.ServiceProxy(serviceURL,
            methods);
        txRresult.value = service.USstatename(1);
    }catch(e){
        printTrace(e);
    }

</script>
```

AJAX e WS: da PHP a JS

Per i veramente pigri: tutto in una sola pagina PHP!

```
<?php
require_once('xmlrpc.inc');
require_once('xmlrpcs.inc');
require_once('ajaxmlrpc.inc');

// definizione delle funzioni e dispatch map: nessuna differenza!

$server = new js_rpc_server($dmap);
?>
<html>
<head>
<?php $server->importMethods2JS(); ?>
</head>
<body>

<a href="#"
    onclick="alert(USstatename(10)); return false;"
>here</a> to execute a webservice call and display results in a
popup message...
```

AJAX e WS: da PHP a JS

Un layout appena piú complesso...

Dispatch map

```
require_once('xmlrpc.inc');  
require_once('xmlrpcs.inc');  
require_once('ajaxxmlrpc.inc');  
  
...
```

Front end: HTML

```
<html><head><?php  
    require('dichiarazioni.inc.php');  
    js_wrap_dispatch_map(...);  
?></head>  
  
...
```

Funzioni esposte

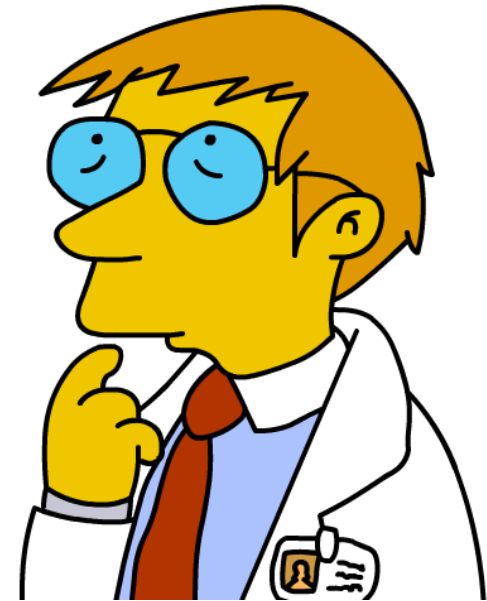
```
<?php  
require('dichiarazioni.inc.php');  
  
...  
?>
```

Front end: WS

```
<?php  
    require('funzioni.inc.php');  
    $server = new xmlrpc_server();  
?>
```

Conclusioni

Domande?



Grazie e arrivederci!