BiPRO-Jubiläum
★ 10 Jahre ★
BiPRO Tag
28./29.06.2016

# BiPRO und PHP

**Marcel Maaß**
MM Newmedia

# Warum eigentlich PHP?


## Weil BiPRO SOAP basiert ist

Das Simple Object Access Protocoll (SOAP) stützt sich auf XML.

- XML ist laut Definition plattform- und implementationsunabhängig. Somit ist jede Programmiersprache, die XML interpretieren kann, für SOAP geeignet.

## Weil PHP mit aktivierter libxml ausgeliefert wird

Die Programmiersprache PHP wird standardmäßig mit aktivierter libxml Erweiterung ausgeliefert. Somit sind alle Voraussetzung für die Interpretation von XML gegeben.

# Bevor wir mit PHP loslegen

## Grundsätzliche Planung und Gedanken

- Objektorientierte Umsetzung der BiPRO Norm, um die Wiederverwendbarkeit der einzelnen Datenobjekte zu erhöhen

  - Keine Verwendung von XML in String Form
    Einfach mal nicht von den gelieferten Beispielen der Provider verwirren lassen

  - XML wird vom SoapClient oder SoapServer automatisch erstellt.
    Die Definition der Datentypen in Form von Objekten ist ausreichend.

  - Grundsatz der Vererbung in einer objektorientierten Umgebung

# Bevor wir mit PHP loslegen


- Gelieferte Daten des Providers überprüfen
  - Ist eine eventuell gelierferte URL zur WSDL Datei überhaupt erreichbar?
  - Sind beispielhafte Requests und Responses vorhanden?
  - Liegen alle XSD Dateien vor?
    Super wichtig, um eventuell von der BiPRO Norm abweichende Datentypen zu ermitteln
  - Ausreichend Kaffee am Start?
    LOS GEHT ‘S!

# Der PHP Soap Client

## Initialisierung

Einfache Initialisierung des Soap Clients

```
$oClient = new \SoapClient(
    $sWsdl,
    [
        'trace' => true,
        'exception' => true,
        'cache_wsdl' => WSDL_CACHE_NONE,
        'compression' => SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP,
        'soap_version' => SOAP_1_2,
    ]
);
```

# Probleme bei der Initialisierung

## Einhaltung von Web Standards

- SOAP-ERROR: Parsing Schema: can’t import schema from ‚http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd‘

  - W3C Delay auf Definitionen / Excessive DTD Traffic
    (Quelle: https://www.w3.org/blog/systeam/2008/02/08/w3c_s_excessive_dtd_traffic/)

  - SoapClient Option “user_agent” ist keine Lösung
    Delay wird trotzdem eingehalten

  - Lösung: WSDL Datei überarbeiten und Definitionen sowie WSDL lokal
    Nachteile: Änderungen seitens des Providers bekommen wir nicht mit

# Weitere zu erwartende Probleme

## Internal Server Error

- Der Provider liefert einfach keine detaillierten Informationen zu Fehlern

  - Hier hilft nur der Kontakt zum Provider und die Auskunft über eventuelle Server Logs seitens des Providers, was ziemlich langwierig und aufwendig ist.

# Datentypen als Objekte

## Benutzt einfach simple Datenobjekte

- Beispiel UsernameToken Knoten für STS

```
class UsernameToken {
    protected $Username;
    protected $Password;

    public function setUsername($sUsername) {
        $this->Username = $sUsername;
    }
    public function setPassword($sPassword) {
        $this->Password = $sPassword;
    }
}
```

- Complex Types sind immer PHP Klassen

# Übergabe an den Soap Client

## Datenobjekte als SoapVar

- Beispiel UsernameToken Knoten für STS

```
$oUsernameTokenVar = new \SoapVar(
    $oUsernameToken,
    SOAP_ENC_OBJECT,
    null,
    'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd',
    'UsernameToken',
    'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd'
);

$oResult = $oClient->RequestSecurityToken($oSoapVar);
```

# Attribute für Datenobjekte

## Beispiel Version Attribut

```
class StatusRequest {
    public $Version = '1.0.0.1'; // Attribut
    public $Status;
}

class Status {
    public $ProzessID = 0;
}
```

### Resultat

```
<ns1:request Version="1.0.0.1">
    <ns1:status>
        <ns1:prozessid>1234</ns1:prozessid>
    </ns1:status>
</ns1:request>
```

# SoapClient mit Attributen

## SoapClient mit Classmap ist die Lösung

```
$oClient = new \SoapClient(
    $sWsdl,
    [
        'trace' => true,
        'exception' => true,
        'classmap' => ['Request' => 'StatusRequest']
    ]
);
$oParams = new StatusRequest();
$oParamsSoapVar = new \SoapVar($oParams, SOAP_ENC_OBJECT, …);
$oClient->__soapCall('getStatus', ['parameters' => ['Request' =>
$oParamsSoapVar]]);
```

# Fazit BiPRO mit PHP


- Provider sollten die W3C Restriktionen beachten
- Lokale WSDL Dateien funktionieren, sind aber nicht das Gelbe vom Ei
- Bei einer objektorientierten Umsetzung ist jeder BiPRO Datentyp ein Objekt
  - Hohe Skalierbarkeit
  - Abweichungen sind durch Vererbung leicht kompensierbar
  - Allein der PHP SoapClient erstellt das XML Schema
- SoapVar und die Classmap Option beachten Attribute anhand der zugewiesenen Namespaces / XSD Definitionen
- Nachteil Geschwindigkeit von SOAP Requests


# BiPRO und PHP

## Vielen Dank für Ihre Aufmerksamkeit.

Marcel Maaß
MM Newmedia

BiPRO
Prozesse | Normen | Community