REST : Codieren der Daten in XML

Das Generieren des XML-Datensatzes ist einfach. Man könnte sogar mit simplen PHP-Stringbefehlen den Code selber erzeugen. Aber es gibt weit elegantere Methoden. Wir verwenden die Spezifikation DOM (Document Object Model). Diese Spezifikation ist in PHP komplett (objektorientiert) integriert. Leider ist die Syntax ein wenig ungewöhnlich, das ist aber auch das einzige Problem.

Referenzieren :

```
$apfel = new DOMDocument(“1.0”)
```

Methode aufrufen :

```
$birne = $apfel -> createElement(“auftrag”)
```

Zu Verfügung stehende Methoden sind unter Anderem :

createElement(“auftrag”)

Damit wird ein Element in XML erzeugt. Also quasi :

<auftrag> …</auftrag>

appendChild($ursprung)

Damit wird ein erzeugtes Element an das Dokument angefügt. Wenn es das Erste ist, nennt man es root-Element.

createTextNode(“2”) ;

Damit kann nun ein Wert an ein Element übergeben werden. Der steht dann in den Element-Tags eingeschlosen.

saveXML()

Das erzeugt einen String, in dem das ganze XML-File enthalten ist. Den kann man z.b. mit print ausgeben, oder bei einer Anwendung mit Apache über cgi ausgeben.

Jetzt ein komplettes Beispiel. Aus einer Datenbank der ERP-Ebene wurde mit PHP auf eine Anfrage hin gelesen, daß "teil1" für ein Produkt aus Variante 2 besteht. Es soll ein XML-Dokument an den aufrufenden Rechner geschickt werden, das einen Auftrag enthält, der aus teil1 besteht, das den Wert 2 hat :

```
<?xml version="1.0"?>
<auftrag>
<teil1>
2
</teil1>
</auftrag>
```

Realisierung in PHP :

```
<?php
$apfel = new DOMDocument("1.0") ;
$ursprung = $apfel -> createElement("auftrag") ;
$apfel -> appendChild($ursprung) ;
   $tag = $apfel -> createElement("teil1") ;
   $ursprung -> appendChild($tag) ;
            $inhalt = $apfel -> createTextNode("2") ;
            $tag -> appendChild($inhalt) ;
print $apfel -> saveXML() ;
$>
```

Erklärung im Einzelnen :

```
$apfel = new DOMDocument(“1.0“) ;
```

$apfel ist Objekt der Klasse DOMDocument

```
$ursprung = $apfel -> createElement(“auftrag”) ;
```

Element $ursprung wird erzeugt, heißt „auftrag“

Methode createElement von $apfel wird aufgerufen

```
$apfel -> appendChild($ursprung) ;
```

$ursprung wird als erstes (=root) Element eingefügt

Methode appendChild von $apfel wird dazu aufgerufen

```
$tag = $apfel -> createElement(“teil1”) ;
```

Element $tag wird erzeugt, heißt „teil1“

Methode createElement von $apfel wird aufgerufen

```
$ursprung -> appendChild($tag) ;
```

$tag wird als Element eingefügt

Methode appendChild von $apfel wird aufgerufen

```
$inhalt = $apfel -> createTextNode(“2”) ;
```

$inhalt wird als Inhalt erzeugt

Methode createTextNode von $apfel wird aufgerufen

```
$tag -> appendChild($inhalt) ;
```

$inhalt wird als Inhalt dem Element $tag zugefügt.

$tag ist selbst ein Objekt, appendChild eine seiner Methoden

```
print $apfel -> saveXML() ;
```

Die Methode saveXML erzeugt das XML-File als String, print gibt es aus.

Ausgabe von XML-Daten auf den Browser

Eine Kleinigkeit muß beachtet werden, wenn man zu Testzwecken REST-Daten auf einem Browser betrachten möchte.
Die Datei, die in der Modellfabrik mit dem Stub get_auftrag.php erzeugt und geschickt wird, sieht so aus (betrachtet mit „Seitenquelltext anzeigen“) :

```
<?xml version="1.0"?>
<auftrag>
<error>0</error>
<teil1>2</teil1>
<teil2>2</teil2>
<teil3>2</teil3>
</auftrag>
```

Die Anzeige des Browsers leider aber so :

```
0222
```

Das liegt daran, daß der Browser eine Information braucht, in welchem Format die Daten daherkommen, die er anzeigen soll. In unserem Fall sind es reine Stringdaten Um dies vernünftig anzuzeigen, sagen wir dem Browser in der zweiten Zeile des Datenstroms : `header("Content-Type: text/plain");`