Manipulating XML with XML::Compile

Colin Bradford

LOVEFILM.COM

The problem

- **XML interface to third party**
  - **Third party validates to a schema**
    - **Only a few elements are optional**
    - **Elements have to be in a particular order**
  - **Different XML elements for different types of requests**
    - **Payment**
    - **Refund**
  - **Different responses depending on what happened**
    - **Error XML**
    - **Response XML**

LOVEFILM.COM

Sample XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
    <Order>
        <OrderID>001-01</OrderID>
        <CardNumber>4444333322221111</CardNumber>
        <Expiry>01/12</Expiry>
        <Amount>100</Amount>
    </Order>
</Request>
```

Solution: print statements

```
print "<?xml version="1.0" encoding="UTF-8"?>\n";
print "<Request>\n";
print "    <Order>\n";
print "        <OrderID>".$order_id."</OrderID>\n";
print "        <CardNumber>".$card_number."</CardNumber>\n";
print "        <Expiry>".$card_expiry."</Expiry>\n";
print "        <Amount>".$amount."</Amount>\n";
print "    </Order>\n";
print "</Request>\n";
```

- Hard to work with
- Inflexible
- Doesn't quote XML entities

LOVEFILM.COM

Solution: XML::LibXML

```
my $doc = XML::LibXML::Document->new();
my $root_element = $doc->createElement('Request');
$doc->setDocumentElement($root_element);
my $order = $doc->createElement("Order");
$root_element->appendChild($order);
$order->appendTextChild('OrderID', $order_id);
$order->appendTextChild('CardNumber', $card_number);
$order->appendTextChild('Expiry', $card_expiry);
$order->appendTextChild('Amount', $amount);
my $xml = $doc->toString();
```

- Full control over XML produced
- Can be verbose and unwieldy

LOVEFILM.COM

Other alternatives

- **XML::Writer**
  - **Easier than XML::LibXML, but still verbose**
- **XML::Simple**
  - **Good if it's structure fits what you need to do, but inflexible**
- **Template Toolkit**
- **and more**

LOVEFILM.COM

XML::Compile

- A CPAN package for Compilation based XML processing
- Takes an XML schema, and produces coderefs to convert between XML and Perl data structures
- Concentrates on correctness and real world usability
  - e.g. hooks to fix schemas that don't match the real world

LOVEFILM.COM

An example XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="Request">
    <xs:complexType>
      <xs:choice>
        <xs:element name="Order" type="orderType"/>
        <xs:element name="Reversal" type="reversalType"/>
      </xs:choice>
    <xs:complexType>
  </xs:element>
  <xs:complexType name="orderType">
    <xs:sequence>
      <xs:element name="OrderID" type="xs:string"/>
      <xs:element name="CardNumber" type="xs:string"/>
      <xs:element name="Expiry" type="xs:string"/>
      <xs:element name="Amount" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="reversalType">
    [snip]
  </xs:complexType>
</xs:schema>
```

LOVEFILM.COM

Using XML::Compile to create XML

- First, discover what the data structure should look like:

```
use XML::Compile::Schema;
my $schema = XML::Compile::Schema->new('request.xsd');
print $schema->template('PERL' => 'Request');
```

LOVEFILM.COM

->template output

```
{ # choice of Order, Reversal
Order =>
  { # sequence of OrderID, CardNumber, Expiry, Amount
    # is a xs:string
    OrderID => "example",
    # is a xs:string
    CardNumber => "example",
    # is a xs:string
    Expiry => "example",
    # is a xs:string
    Amount => "example", },
  # is a reversalType
  Reversal => { [ snip ] }
}
```

LOVEFILM.COM

Create a hash in the right format

```
my $hash = {
Order =>
   {

     OrderID => "001-01",

     CardNumber => "4444333322221111",

     Expiry => "01/12",

     Amount => "1.00", },


}
```

LOVEFILM.COM

and pass the hash to XML::Compile

```
my $schema = XML::Compile::Schema->new('request.xsd');
my $doc = XML::LibXML::Document->new('1.0', 'UTF-8');
my $writer = $schema->compile(WRITER => 'Request');
my $xml = $writer->($doc, $hash);
$doc->setDocumentElement($xml);
print $doc->toString();
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Order>
    <OrderID>001-01</OrderID>
    <CardNumber>4444333322221111</CardNumber>
    <Expiry>01/12</Expiry>
    <Amount>100.00</Amount>
  </Order>
</Request>
```

LOVEFILM.COM

Reverse direction: XML to Perl

- **Response looks like:**

```
<Response>
    <OrderResp>
        <OrderID>001-01</OrderID>
        <TxRefNum>Tx123456</TxRefNum>
        <ApprovalStatus>1</ApprovalStatus>
        <StatusMsg>Approved</StatusMsg>
    </OrderResp>
</Response>
```

LOVEFILM.COM

Response schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Response">
    <xs:complexType>
      <xs:choice>
        <xs:element name="OrderResp" type="orderRespType"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="orderRespType">
    <xs:sequence>
      <xs:element name="OrderID" type="xs:string"/>
      <xs:element name="TxRefNum" type="xs:string"/>
      <xs:element name="ApprovalStatus" type="xs:string"/>
      <xs:element name="StatusMsg" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

LOVEFILM.COM

Create a reader, and pass it the data

```
my $schema = XML::Compile::Schema->new('response.xsd');
my $reader = $schema->compile(READER => 'Response');
my $hash = $reader->("response.xml");

say Data::Dump::pp($hash);
```

```
{
  OrderResp => {
        ApprovalStatus => 1,
        OrderID        => "001-01",
        StatusMsg      => "Approved",
        TxRefNum       => "Tx123456",
      },
}
```

LOVEFILM.COM

Comments and caveats

- **Schema must match the data - otherwise XML::Compile (correctly) complains**
- **Don't assume that third parties actually follow their own schemas fully**
- **Creating the coderefs is slow - cache where possible**
- **The module author (Mark Overmeer) is really responsive and helpful**

Thank you

Colin Bradford