Streaming XML

Rob Richards

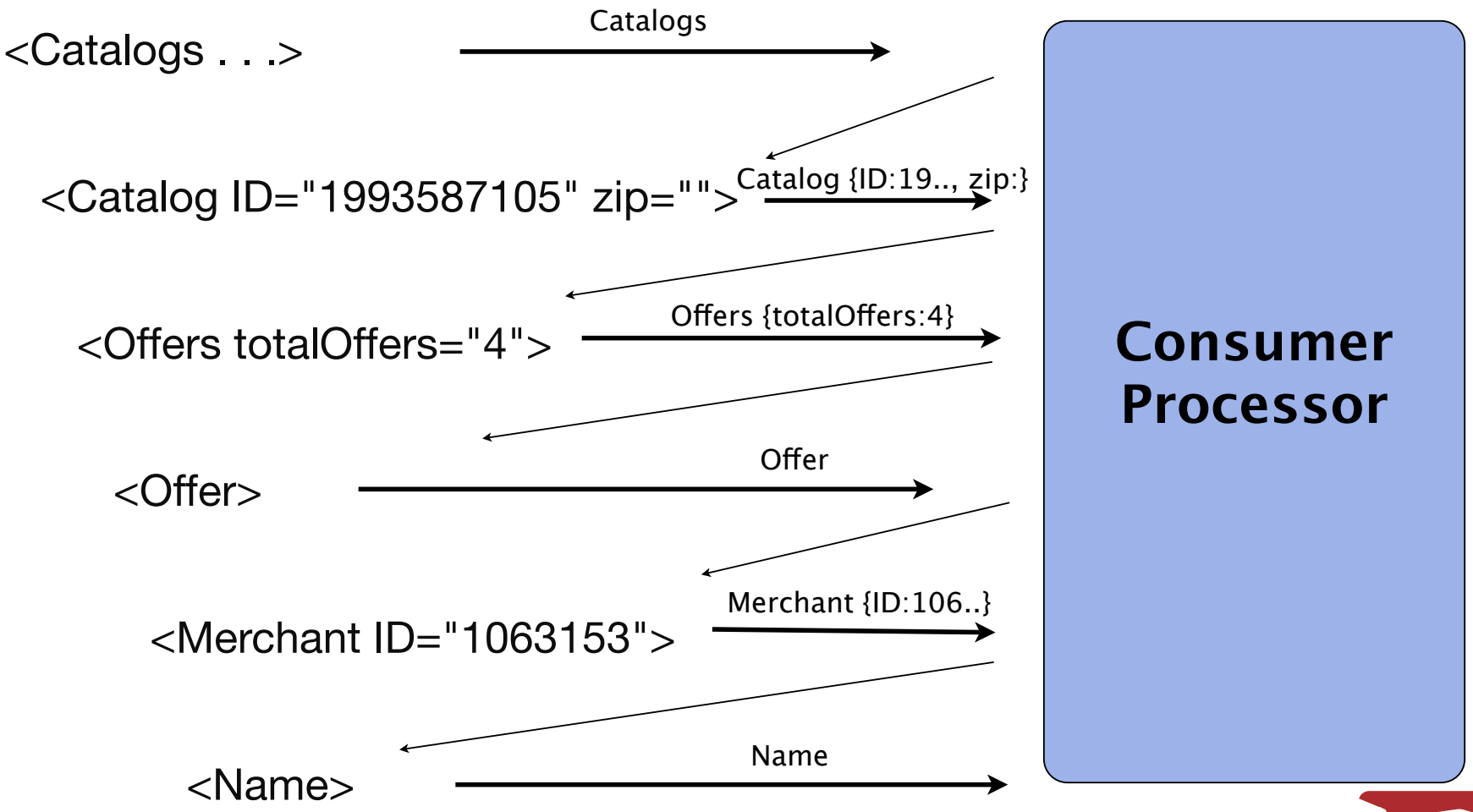
May 20, 2009

http://www.cdatazone.org
http://xri.net/=rob.richards

Parsing XML

- Push Parser
  - Parser scans document and emits events to execute caller's callbacks
- Tree Parser
  - XML is read into memory and converted into a Tree
- Pull Parser
  - XML data is processed as needed and determined by the caller

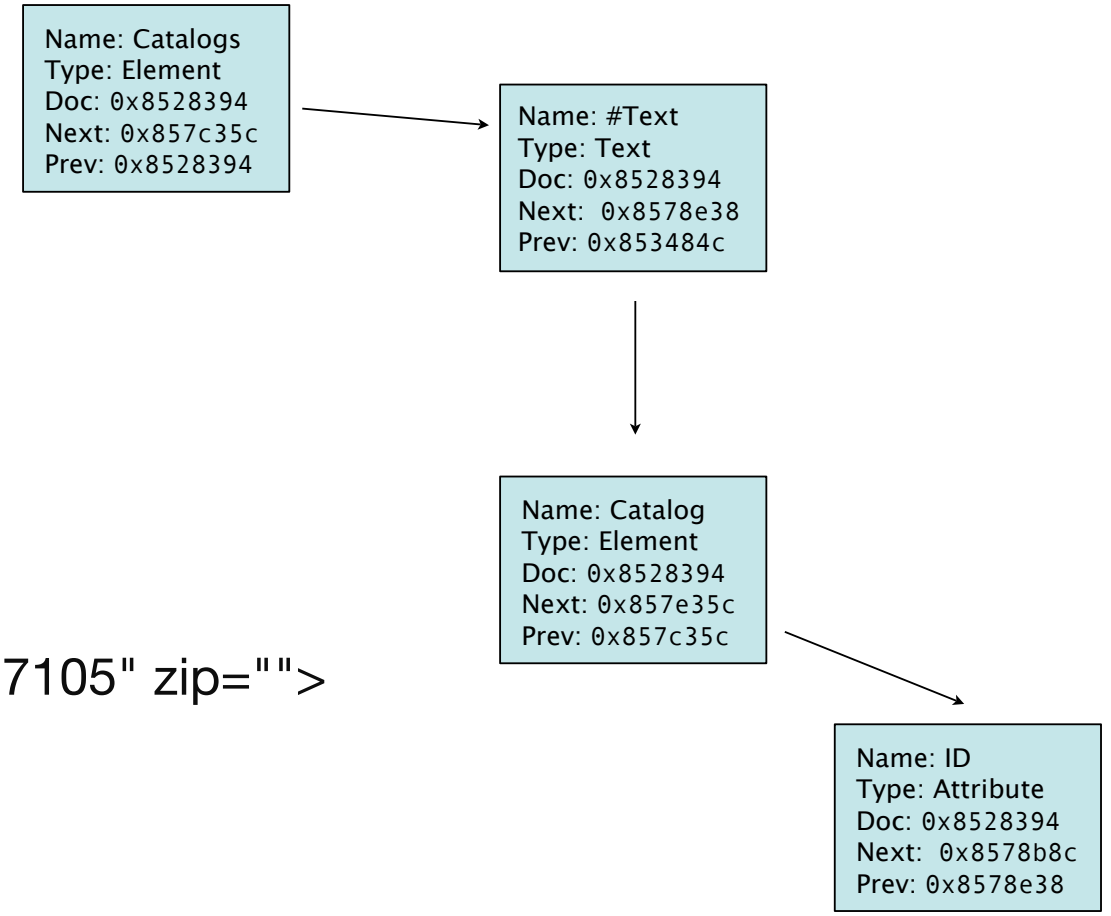
Push Parser



MASHERY

Tree Parser

<Catalogs . . .>



<Catalog ID="1993587105" zip="">

MASHERY

SAX (ext/xml)

- Event based push parser
- Works using function callbacks
- Low memory usage
- Read-only
- Forward only
- No validation support

ext/xml Example

```
function startElement($parser, $name, $attrs) {
    // start element handler logic
}

function endElement($parser, $name) {
    // end element handler logic
}

$xml_parser = xml_parser_create();
xml_set_element_handler($xml_parser, "startElement", "endElement");

while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp)))
        echo "ERROR!";
}
xml_parser_free($xml_parser);
```

DOM

- Tree based parser
- Allows for creation and editing of XML documents
- Provides XPath and XInclude Support
- Provides validation support
  - DTD
  - XML Schemas
  - RelaxNG
- Ability to work with HTML documents
- Zero copy interoperability with SimpleXML

DOM Example

```
$dom = new DOMDocument();
$dom->load(<URI>);

$root = $dom->documentElement;
foreach ($root->childNodes AS $node) {
    if ($node->nodeType == XML_ELEMENT_NODE) {
        echo $node->nodeName;
    }
}

$child = $root->firstChild;
$parent = $child->parentNode;
```

SimpleXML

- Tree based parser
- Provides simple access to XML documents
- Operates only on elements and attributes
- Contains XPath support
- Allows for modifications to the XML
- Zero copy interoperability with DOM

SimpleXML Example

```
$sxe = simplexml_load_string('<root/>');
$sxe->child = '';
$sxe->child->nodea = 'A';
$sxe->child->nodeb = 'B';
$nodea = $sxe->child->nodea;

$nodes = $nodea->xpath('/root/child');

$childnode = $nodes[0];

echo $childnode->nodea;

$node = dom_import_simplexml($childnode);
echo $node->nodeName;
```


MASHERY

XMLReader

- It is a Pull parser
- Based on the C# XmlTextReader API
- Forward moving stream based parser
- Advantages
  - Low memory footprint
  - Simple API
  - Faster Processing
  - Namespace support
  - Validation support
  - Advanced Feature Set

Simple XML Document

```
<root>
    <child cattr="123">my content</child>
    <?php echo 'hello world';?>
    <![CDATA[ random data ]]>
</root>
```


MASHERY

Simple Parsing

```
$reader = new XMLReader();
$reader->XML($xml);
while ($reader->read())
{
        echo "Name: " . $reader->name."\t";
        echo "Value: " . $reader->value."\n";
}
$reader->close();
```


MASHERY

Simple Parsing Results

Name: root Value:
Name: #text Value:

Name: child Value:
Name: #text Value: my content
Name: child Value:
Name: #text Value:

Name: php Value: echo 'hello world';
Name: #text Value:

Name: #cdata-section Value: random data
Name: #text Value:

Name: root Value:

Reading Data

- XML($stringInput [, $encoding [, $options]])
  - Reads and XML document loaded within a string

- open($URI [, $encoding [, $options]])
  - Reads an XML document at location specified by URI
  - Uses PHP streams to read the data

MASHERY

Working With XML

```
$url = 'http://developer.ebay.com/webservices/latest/eBaySvc.wsdl';

$begin = microtime(true);
$sxe = simplexml_load_file($url);
$end = microtime(true);
echo "Root node: " . $sxe->getName() ."\n";
$total = $end - $begin;
echo "Elapsed time: " . $total."\n";
```

Working With XML

Root node: definitions

Elapsed time: 20.740789175

Streams To The Rescue

```
$url = 'http://developer.ebay.com/webservices/latest/eBaySvc.wsdl';
$begin = microtime(true);

$reader = new XMLReader();
$reader->open($url);
while ($reader->read()) {
        if ($reader->nodeType == XMLReader::ELEMENT) {
                echo "Root node: " . $reader->localName ."\n";
                break;
        }
}
$reader->close();
$end = microtime(true);
$total = $end - $begin;
echo "Elapsed time: " . $total."\n";
```


MASHERY

Streams To The Rescue

Root node: definitions
Elapsed time: 0.236433029175

100X Faster!

Memory Impact

```
echo "Memory used: " . memory_get_usage()."\n";
```

SimpleXML Example

Memory used: 64000

XMLReader Example

Memory used: 65448

MASHERY

Real Memory Impact

```
echo "Memory used: " . memory_get_usage()."\n";
```

Debug build of libxml2 used to capture its maximum memory usage

SimpleXML Example

Memory used: 64000

libxml2 memory usage: 12980964 (**13 MB**)

XMLReader Example

Memory used: 65448

libxml2 memory usage: 28098 (**28 KB**)

MASHERY

Node Types

XMLReader::ELEMENT	Element opening tag
XMLReader::END_ELEMENT	Element closing tag
XMLReader::ATTRIBUTE	Attribute
XMLReader::TEXT	Text node
XMLReader::CDATA	CDATA node
XMLReader::PI	Processing Instruction
XMLReader::COMMENT	Comment Node
XMLReader::WHITESPACE	Whitespace
XMLReader::NONE	No Data: BOF or EOF

Node Types

```
while ($reader->read()) {
        switch ($reader->nodeType) {

                case XMLReader::ELEMENT:
                        echo '<' . $reader->localName . '>';
                        break;

                case XMLReader::END_ELEMENT:
                        echo '</' . $reader->localName . '>';
                        break;

                case XMLReader::TEXT:
                case XMLReader::SIGNIFICANT_WHITESPACE:
                        echo $reader->value;
        }
}
```


MASHERY

Attributes

```
<root>
        <child attr1="abc" attr2="def" />
        <child xmlns:pfx="urn::pfx" pfx:attr1="123" />
</root>

/* Position cursor at first child element */
while ($reader->read()) {
        if ($reader->localName == 'child') {
                /* insert code here */
                break;
        }
}
```


MASHERY

Attributes

```
<root>
        <child attr1="abc" attr2="def" />
        <child xmlns:pfx="urn::pfx" pfx:attr1="123" />
</root>
```

```
echo 'Attr1: ' . $reader->getAttribute("attr1") . "\n";          // Attr1: abc
echo 'Attr2: ' . $reader->getAttribute("attr2") . "\n";          // Attr2: def

$reader->next('child');
echo 'Attr1: ' . $reader->getAttribute("attr1") . "\n";          // Attr1:
echo 'Attr1: ' . $reader->getAttributeNS("attr1",
                                         "urn::pfx") . "\n";     // Attr1: 123

echo 'Attr1: ' . $reader->getAttributeNo(1) . "\n";              // Attr1: 123
echo 'Attr1: ' . $reader->getAttributeNo(0) . "\n";              // Attr1: urn::pfx
```


MASHERY

Attributes

```
<root>
        <child attr1="abc" attr2="def" />
        <child xmlns:pfx="urn::pfx" pfx:attr1="123" />
</root>
```

```
$reader->moveToFirstAttribute();
echo $reader->name . ': ' . $reader->value."\n";                // attr1: abc
while ($reader->moveToNextAttribute()) {
            echo $reader->name . ': ' . $reader->value."\n";    // attr2: def
}

$reader->moveToElement();
$reader->next('child');
$reader->moveToAttributeNo(0);
echo $reader->name . ': ' . $reader->value."\n";        // xmlns:pfx: urn::pfx
```


MASHERY

XMLReader Properties

- name
- localName
- nodeType
- hasValue
- value
- hasAttributes
- attributeCount
- depth
- prefix
- namespaceURI
- baseURI
- isDefault
- isEmptyElement
- xmlLang

Basic Parser Configuration (Depreciated)

- XMLReader::LOADDTD
- XMLReader::DEFAULTATTRS
- XMLReader::VALIDATE
- XMLReader::SUBST_ENTITIES

```
$reader = newXMLReader();
$reader->open($file);
$reader->setParserProperty(XMLReader::LOADDTD, TRUE);
$reader->setParserProperty(XMLReader::VALIDATE, TRUE);
var_dump($reader->getParserProperty(XMLReader::VALIDATE));
```


MASHERY

Advanced Parser Configuration

- Leverages ext/libxml parser constants
- Provides advanced parsing capabilities to XMLReader

```
$reader = new XMLReader();
$reader->XML($xml, NULL,
        LIBXML_NOCDATA | LIBXML_DTDVALID | LIBXML_XINCLUDE);

echo $reader->nodeType."\n";
while ($reader->read()) {
        echo $reader->name."\n";
}
```


MASHERY

DTD Validation

```
<!DOCTYPE chapter [
    <!ELEMENT chapter (title, para)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT para (#PCDATA)>
]> <chapter>   <title>XMLReader</title>   </chapter>

$reader->XML($xml, NULL, LIBXML_DTDVALID | LIBXML_NOERROR);
while ($reader->read()) {
        echo $reader->name . ': ' . $reader->value."\n";
        if (! $reader->isValid()) {
                echo libxml_get_last_error()->message;
                break;
        }
}
```

chapter:
Element chapter content does not follow the DTD, expecting (title , para), got (title)

MASHERY

RelaxNG Validation

XML Source

```
<chapter>
        <title>XMLReader</title>
        <test/>
</chapter>
```

RelaxNG Source

```
<element name="chapter" xmlns="http://relaxng.org/ns/structure/1.0">
  <element name="title"><text/></element>
  <element name="para"><text/></element>
</element>
```


MASHERY

RelaxNG Validation

```
$reader = new XMLReader();
libxml_use_internal_errors(true);
$reader->XML($xml);
$reader->setRelaxNGSchemaSource($rng);

while ($reader->read()) {
        if (! $reader->isValid()) {
                echo $reader->name . ': ' . $reader->value."\n";
                echo libxml_get_last_error()->message;
                break;
        }
}
```

test:
Did not expect element test there

MASHERY

XML Schema Validation

```
<chapter>
        <title>XMLReader</title>
        <test/>
</chapter>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="chapter">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" minOccurs="1">
          <xsd:complexType>    <xsd:sequence>
              <xsd:element name="title"  minOccurs="1" type="xsd:string"/>
              <xsd:element name="para"  minOccurs="1" type="xsd:string"/>
            </xsd:sequence>     </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```


MASHERY

XML Schema Validation

```
$reader = new XMLReader();
libxml_use_internal_errors(true);
$reader->XML($xml);
$reader->setSchema('schema.xsd');

while ($reader->read()) {
        if (! $reader->isValid()) {
                echo $reader->name . ': ' . $reader->value."\n";
                echo libxml_get_last_error()->message;
                break;
        }
}
```

chapter:
Element 'test': This element is not expected.

MASHERY

Exporting Nodes

```
<movie>
    <title>PHP: Behind the Parser</title>
    <character>
        <name>Ms. Coder</name>
    </character>
</movie>

while($reader->read()) {
            if ($reader->name == 'title') {
                        $node = $reader->expand();
                        $dom = new DOMDocument();
                        $node = $dom->importNode($node, true);
                        print $dom->saveXML($node);
                        break;
            }
}
```

<title>PHP: Behind the Parser</title>

MASHERY

XMLWriter

- Lightweight, forward-only API for generating well formed XML
  - Automatically escapes data
  - Writes to memory or directly to streams
  - Ability to control indenting
  - Enforces wellformedness by automatically closing open tags

Creating The Document

```
$writer = new XMLWriter();
$writer->openMemory();

$writer->setIndent(true);
$writer->setIndentString("\t");
$writer->startDocument('1.0', 'UTF-8');

$writer->startElement('results');

echo $writer->flush(false);
```

```
<?xml version="1.0" encoding="UTF-8"?>
<results
```


MASHERY

Writing Content

```
$writer->startElement('result');
$writer->writeAttribute("id", "123");
$writer->writeElement('name', 'John & Jane');
$writer->startElement('lastName');
$writer->text('Doe');
$writer->endElement();

echo $writer->flush()
```

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
    <result id="123">
        <name>John &amp;amp; Jane</name>
        <lastName>Doe</lastName>
```


MASHERY

Completing The Document

```
$writer->endElement();
$writer->startElement('result');
$writer->writeAttribute("id", "456");
$writer->startElement('name');
$writer->writeRaw('Joe &amp; Mary');
$writer->endElement();
$writer->writeElement('lastName', 'Smith');
$writer->endDocument();
echo $writer->flush()."\n";
```

Add raw content

Close off all open tags

```
        </result>
        <result id="456">
            <name>Joe &amp; Mary</name>
            <lastName>Smith</lastName>
        </result>
</results>
```


MASHERY

Writing To Streams

- Requires less resources when outputting XML
  - Faster web service response time
  - Less server load
  - Write directly to php://output
  - Write to custom stream wrappers
  - Automatic buffer flushing as needed

Writing To Streams

```
$data = array(array('name' => 'Dick & Jane', 'number' =>123),
array('name' => 'John & Mary', 'number' => 456));

$writer = new XMLWriter();
$writer->openURI('php://output');
$writer->setIndent(true);

$writer->startElement('results');
foreach ($data AS $record) {
        $writer->startElement('result');
        foreach ($record AS $key => $value) {
                $writer->writeElement($key, $value);
        }
        $writer->endElement();
        $writer->flush();
}

$writer->endDocument();
unset($writer);
```


MASHERY

Writing To Streams

```
<results>
 <result>
  <name>Dick &amp;amp; Jane</name>
  <number>123</number>
 </result>
 <result>
  <name>John &amp;amp; Mary</name>
  <number>456</number>
 </result>
</results>
```

Namespaces: Brute Force

```
$writer = new XMLWriter();
$writer->openMemory();
$writer->setIndent(true);

$writer->startElement('pfx:root');
$writer->writeAttribute('xmlns:pfx', 'urn::mypfx');
$writer->writeElement('pfx:child', 'mydata');

$writer->endDocument();
echo $writer->flush();
```

```
<pfx:root xmlns:pfx="urn::mypfx">
 <pfx:child>mydata</pfx:child>
</pfx:root>
```


MASHERY

Namespaces

```
$writer = new XMLWriter();
$writer->openMemory();
$writer->setIndent(true);

$writer->startElementNS('pfx', 'root', 'urn:mypfx');
$writer->writeElementNS('pfx', 'child', 'urn:mypfx', 'mydata');

$writer->endDocument();
echo $writer->flush();
```

```
<pfx:root xmlns:pfx="urn:mypfx">
 <pfx:child xmlns:pfx="urn:mypfx">mydata</pfx:child>
</pfx:root>
```

MASHERY

Streaming XML

Rob Richards

http://www.cdatazone.org
http://xri.net/=rob.richards