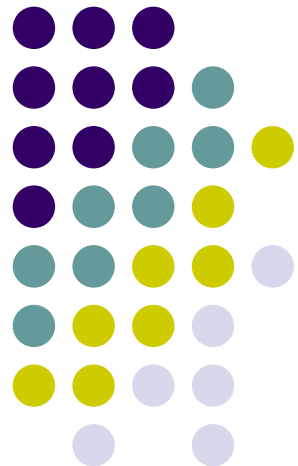
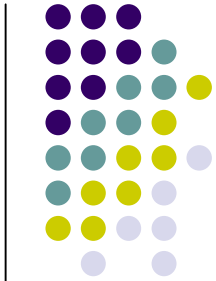
PHP & XML – PHP & Web services

Presenters:
Sonu
Nathan
Chris
Mansoor

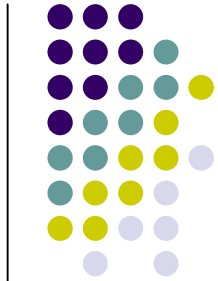


Topic 1 -
PHP & Web Services



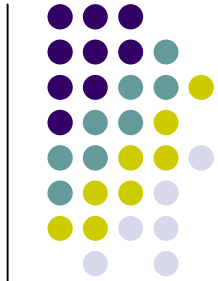
- Why PHP
  - Easily usable packages
  - Simplified interface and details
  - Looks and feels like Perl
- Major SOAP Packages available
  - NuSOAP
  - PHP:PEAR::SOAP
- This tutorial is based on NuSOAP

Server using NuSOAP



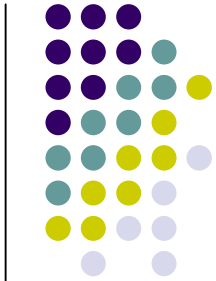
- Download @: http://dietrich.ganx4.com/nusoap/index.php
- Has built-in WSDL support
  - No document descriptor XML file required
  - WSDL file can be loaded and registered on the fly with very few lines of code

Real Live Example



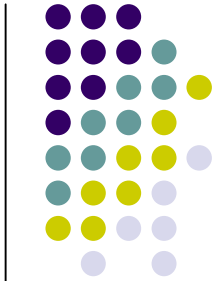
- The following code is an example of a server built in PHP that returns the GST for a given amount
- It uses the nusoap.php package and assumes that the web server is configured to handle PHP scripts

The Server Code



```
1. <?
2. require_once("nusoap.php");
3. $ns="www.yourserver.com/";
4. $server = new soap_server();
5. $server->configureWSDL('CanadaTaxCalculator',$ns);
6. $server->wsdl->schemaTargetNamespace=$ns;
7. $server->register('CalculateTax',
8.                   array('amount' => 'xsd:string'),
9.                   array('return'=>'xsd:string'),
10.                  $ns);
11. function CalculateTax($amount){
12.                  $taxcalc=$amount*.07;
13.                  return new soapval('return','xsd:string',$taxcalc);
14. }
15. $server->service($HTTP_RAW_POST_DATA);
16. ?>
```

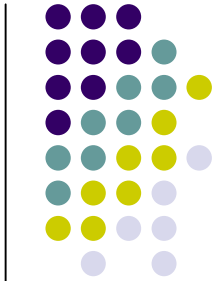
The Server Code - Explained



- 1. <?
- …….
- …….
- 16. ?>

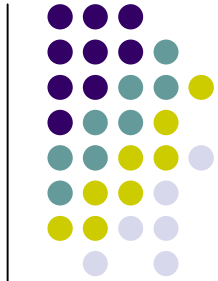
- Line 1 and 16 denote the start and end of a PHP script.

The Server Code - Explained



- 2. require_once("nusoap.php");
- Line 2 includes the NuSOAP package
- 3. $ns=“www.yourserver.com/";
- Line 3 designates the URI for the web service

The Server Code - Explained



- 4. $server = new soap_server();

- Line 4 create a new instance of a soap_server object

- 5. $server->configureWSDL('CanadaTaxCalculator',$ns);
- 6. $server->wsdl->schemaTargetNamespace=$ns;

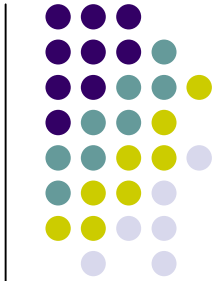
- Lines 5 and 6 configure the service name and namespace for the WSDL

The Server Code - Explained

- 7. $server->register('CalculateTax',
- 8. array('amount' => 'xsd:string'),
- 9. array('return'=>'xsd:string'),
- 10. $ns);

- Lines 7-10 make the server aware of the function/method ‘CalculateTax’ which takes in a string and returns a string

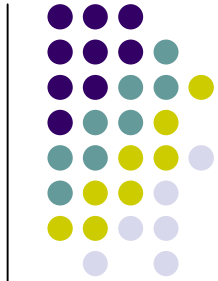
The Server Code - Explained



```
11. function CalculateTax ($amount) {
12.             $taxcalc = $amount * .07;
13.             return new soapval ('return', ‘xsd:string', $taxcalc);
14. }
```

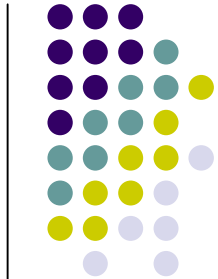
- Lines 11-14 define the ‘CalculateTax’ method.
- Notice the return value

The Server Code - Explained



- 15. $server->service($HTTP_RAW_POST_DATA);
- Line 15 simply invokes the service
- And Voila!
- The above code needs to be placed on the web server, and the service is now live

Auto Generated WSDL

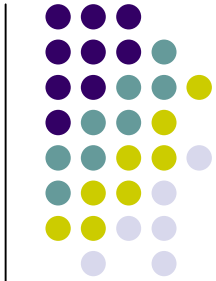


- One of NuSOAP's strengths is the built in WSDL generation capability
- With any servers built using NuSOAP and PHP, adding "?wsdl" to the end of the server's URL will dynamically generate and display the WSDL
- http://www.yourserver.com/server.php?wsdl

Client using NuSOAP

- NuSOAP’s built-in WSDL support simplifies the client creation
- 1. <?
- 2. require_once('nusoap.php');
- 3. $wsdl = “http://www.yourserver.com/server.php?wsdl”;
- 4. $client=new soapclient($wsdl, 'wsdl');
- 5. $param=array('amount'=>'15.00',);
- 6. echo $client->call('CalculateTax', $param);
- 7. ?>

Client code - Explained

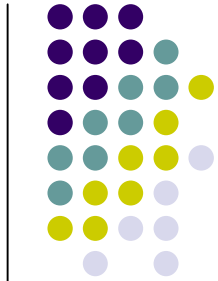


- NuSOAP's built-in WSDL support simplifies the client creation

- 3. $wsdl = "http://www.yourserver.com/server.php?wsdl";

- Line 3 assigns a variable to a WSDL.
  - **Note:** the WSDL is dynamically generated by appending "?wsdl" to the server's URL

Client using NuSOAP



- NuSOAP’s built-in WSDL support simplifies the client creation

- 4. $client=new soapclient($wsdl, 'wsdl');

- Line 4 creates a new instance of a soap client with the WSDL defined earlier

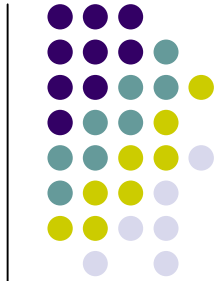
Client using NuSOAP

- NuSOAP’s built-in WSDL support simplifies the client creation

- 5. $param=array('amount'=>'15.00',);

- Line 5 creates the parameter to be passed to the web service
  - In this example, the client is requesting the web service to return the GST value for a $15.00 purchase
  - This value can be dynamic

Client using NuSOAP



- NuSOAP’s built-in WSDL support simplifies the client creation

- 6. echo $client->call('CalculateTax', $param);

- Line 6 makes a call to the web service and displays the return result via the ‘echo’ call

    - And you’re done!

Topic 2 -
XML Parsing in PHP 5 – Dom Model

- PHP 5 has a DOM extension that fully conforms to the W3C standards

- Compatibility: In following excellent Microsoft standards, PHP 5 is completely incompatible with PHP 4 and below
  - Note: The Microsoft reference is completely wrong ☺

Dom Model – Step by Step Guide Creation

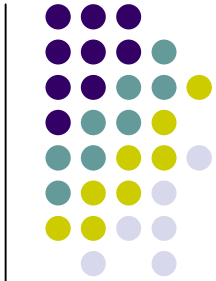
1. Create a new DomDocument object

- $dom = new DomDocument();

2. Load an XML file:

- $dom->load("articles.xml");

Dom Model – Step by Step Guide Validation

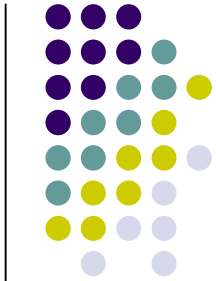


3. Validation can be done for DTDs, XML Schemas, and RelaxNG documents using the appropriate validate method:

- $dom->validate("mydtd.dtd");
- $dom->schemaValidate("myschema.xsd");
- $dom->relaxNGValidate("myRNG.rng");

- Note: The return value is a boolean, with any errors returned as PHP warnings

Dom Model – Step by Step Guide Element Access



4. DomDocument objects can be traversed in two ways:

   4.a Getting a list of all nodes

   ```
   $mytags = $dom->getElementsByTagName("myTag");
   ```

   4.b Getting a unique node

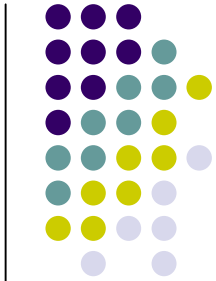
   ```
   $myID = $dom->getElementByID("myID");
   ```

Dom Model – Step by Step Guide Children Access

5. Children nodes for a given node can be accessed as follows:

- $children = $myNode->childNodes;

Dom Model – Step by Step Guide Dynamic XML Creation



- DomDocuments can also write out XML files
- This functionality can be used to easily create/modify XML Documents

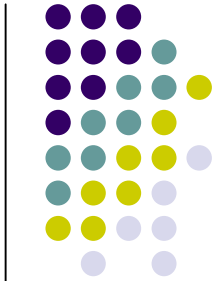
Dom Model – Step by Step Guide Dynamic XML Creation - Example

- Adding new elements to an XML document

- $myElement = $dom->createElement("myElement");
- $myText = $dom->createTextNode("A Text Node");
- $myElement->appendChild($myText);
- $dom->documentElement->appendChild($myElement);

- **Note:** The nodes are created and chained together, then the new element is inserted into the root element

Dom Model – Step by Step Guide Dynamic XML Creation - Example

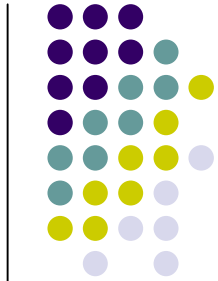


- XML Documents can be generated in two ways

1. Output the document to a web browser, or standard output
   - print $dom->saveXML();

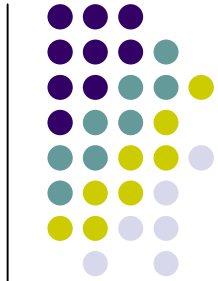
2. Output the document to file
   - print $dom->save("myXMLfile.xml");

Dom Model in PHP – Benefits



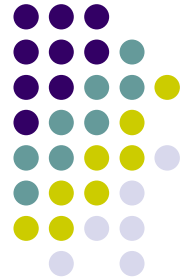
- Ease of use
- Provided a simple interface for a multitude of tasks including validation, querying and modifying XML
- In memory parsing implies fast, non-sequential access
- Handy for visualizing data and transforming data on the fly

Dom Model in PHP – Drawbacks



- Parallel parsing of large documents will hog system memory
- Building the in-memory tree takes a long time
- Does not support partial parsing of XML documents

Topic 3 -
XML Parsing in PHP 5 – Event Based

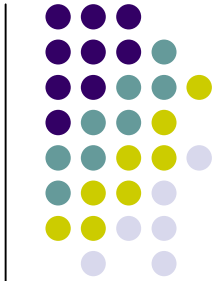


- This example is based on RSS (Really Simple Syndication)

- RSS: document format intended to describe, summarize, and distribute the contents of a Web site as a ‘channel’ (used for sending news headlines and other contents by BBC, CNN, Disney, Slashdot…)

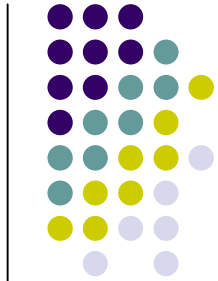
- Channel: The aggregation of various documents (pages) on a site

XML Parsing – Event Based Example



- http://ugweb.cs.ualberta.ca/~muhammad/RSS/rss.xml
- Most browsers display XML as is – or worse
- Need a better mechanism that can selectively parse and display the file
- DOM models are not the best option since they parse the entire document resulting in excessive, unneeded overhead

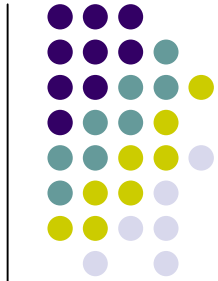
XML Parsing – Event Based Example



- We use an intermediate agent based on PHP and event based parsing that formats all XML documents in a channel

- This example is Object Oriented

XML Parsing – Event Based Example

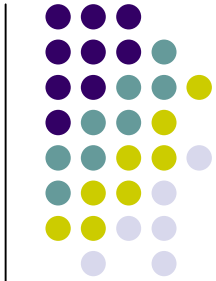


- Create a class RSSParser

- class RSSParser {
  var $insideitem = false;
  var $tag = "";
  var $title = "";
  var $description = "";
  var $link = "";

- The above variables hold all the information we will need since we do not need to remember extraneous tags

XML Parsing – Event Based Example



- There are 3 functions within this class

- ```
  function startElement($parser, $tagName, $attrs) {
      if ($this->insideitem) {
          $this->tag = $tagName;
      }
      elseif ($tagName == "ITEM") {
          $this->insideitem = true;
      }
  }
  ```

- startElement is called each time we see a tag while reading the input XML.
- Notice how it ignores all tags except the “ITEM” tag
  ```

XML Parsing – Event Based Example

- function characterData($parser, $data) {
  if ($this->insideitem) {
  switch ($this->tag) {
  case "TITLE":
  $this->title .= $data;
  break;
  case "DESCRIPTION":
  $this->description .= $data;
  break;
  case "LINK":
  $this->link .= $data;
  break;
  }
  }
  }

$this – This parser

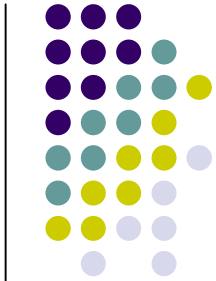
insideitem – set to true when inside a "ITEM" tag

The "ITEM" tag contains these three tags

$data – the data pertaining to the current tag

- Gets called one or more times in response to the event of reaching text within a set of tags

XML Parsing – Event Based Example

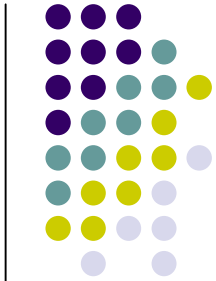


- function endElement($parser, $tagName) {
    if ($tagName == "ITEM") {
      printf(" %s ",
      trim($this->link),htmlspecialchars(trim($this->title)));
      printf(" %s ",
      htmlspecialchars(trim($this->description)));
      $this->title = "";
      $this->description = "";
      $this->link = "";
      $this->insideitem = false;
    }
  }
  }

This brace close the initial brace of the class declaration: class RSSParser {

- Called when a tag is closed, If the tag == “ITEM”, our initially declared variables hold all the data we need to display the contents of this “ITEM” tag

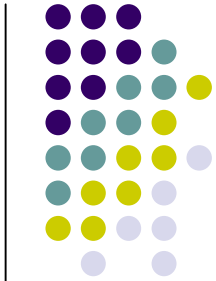
XML Parsing – Event Based Example



- Now we can use the above RSS Parser class to feed and output all “ITEMS” within an XML file

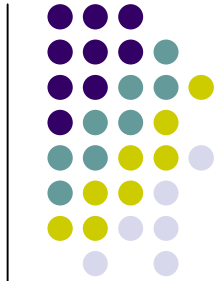
```
1. $xml_parser = xml_parser_create();  //php built in event-driven xml parser
2. $rss_parser = new RSSParser();
3. xml_set_object($xml_parser,&$rss_parser);
//set the event handlers for the start and end tags of an element
4. xml_set_element_handler($xml_parser, "startElement", "endElement");
//event handler for the text data between inside tags
5. xml_set_character_data_handler($xml_parser, "characterData");
6. $fp = fopen("http://www.sitepoint.com/rss.php","r")
  or die("Error reading RSS data.");
7. while ($data = fread($fp, 4096)) //read in 4K chunks
  8. xml_parse($xml_parser, $data, feof($fp))
    or die(sprintf("XML error: %s at line %d",
      xml_error_string(xml_get_error_code($xml_parser)),
      xml_get_current_line_number($xml_parser)));
9. fclose($fp);
10. xml_parser_free($xml_parser);
```

XML Parsing – Event Based Example



- 1. $xml_parser = xml_parser_create();
- Line 1 Creates an instance of the built in event-driven xml parser
- 2. $rss_parser = new RSSParser();
- Line 2 creates an instance of our class

XML Parsing – Event Based Example



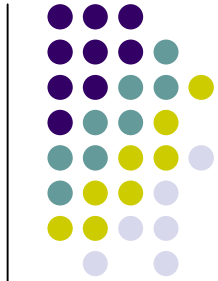
- 3. xml_set_object($xml_parser,&$rss_parser);

- Line 3 ties our object as the event handler for the XML parser

- 4. xml_set_element_handler($xml_parser, "startElement", "endElement");

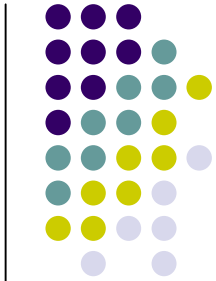
- Line 4 registers the individual functions that get fired when the parser sees a start/end tag

XML Parsing – Event Based Example



- 5. xml_set_character_data_handler($xml_parser, "characterData");
- Line 5 is the event handler for the text data between inside
- 6. $fp = fopen("http://www.sitepoint.com/rss.php","r")
  or die("Error reading RSS data.");
- Line 6 opens the RSS feed
- 7. while ($data = fread($fp, 4096))
- Line 7 reads in data in 4KB (or less for the last one) chunks

XML Parsing – Event Based Example

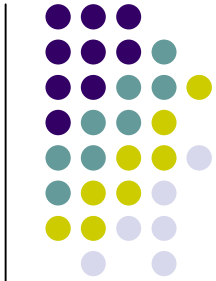


- 8. While loop:

- xml_parse($xml_parser, $data, feof($fp))
  or die(sprintf("XML error: %s at line %d",
  xml_error_string(xml_get_error_code($xml_parser)),
  xml_get_current_line_number($xml_parser)));

- The while loop parsing each chunk of the data

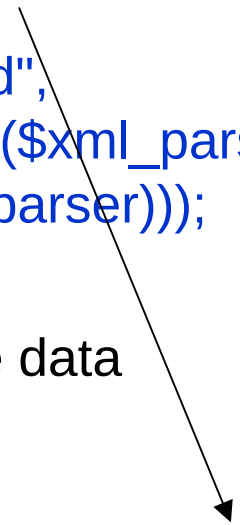
XML Parsing – Event Based Example



- 8. While loop:

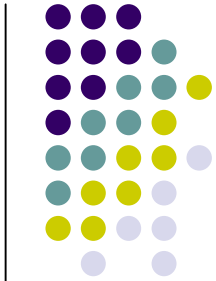
- xml_parse($xml_parser, $data, feof($fp))
  or die(sprintf("XML error: %s at line %d",
  xml_error_string(xml_get_error_code($xml_parser)),
  xml_get_current_line_number($xml_parser)));

- The while loop parsing each chunk of the data



This is where the events get fired (the functions are called)

XML Parsing – Event Based Example



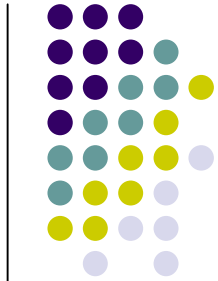
- 8. While loop:

- xml_parse($xml_parser, $data, feof($fp))
  or die(sprintf("XML error: %s at line %d",
  xml_error_string(xml_get_error_code($xml_parser)),
  xml_get_current_line_number($xml_parser)));

- The while loop parsing each chunk of the data

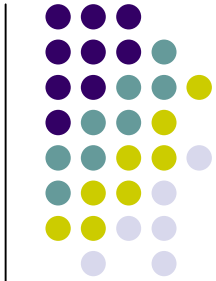
- 9. fclose($fp); → Close the file
  10. xml_parser_free($xml_parser); → Free the memory

XML Parsing – Event Based Benefits



- Easy readability of RSS XML feeds
- Overall beneficial when parsing chunks of documents
- Extremely tight on memory usage, especially compared to the DOM model
- As easy as 1-2-3: Create the parser, set the handlers and pass the feed

XML Parsing – Event Based Drawbacks



- No advanced functionality of the DOM model, such as non-sequential access
- Complex searches can be difficult to implement
- No DTD available
- Lexical information is not available