Parsing, Validating and Saving Data from Complex XML Streams:
Lessons Learned While Developing ST Parser

New York PHP Monthly Meeting
August 26, 2003

by Daniel Convissor
The Analysis and Solutions Company
Introduction

- Project background
- Explain a bit about XML
- Quick points regarding various parsers
- Demonstration of using PHP's SAX based parser
- Validation

Please Note:

- Sample code has been significantly simplified
- If viewing on the web: set browser to full screen mode
- Opera 7 Win32 users: our style sheet causes rendering delays, even though it's valid
- Footer placement works right in Mozilla on multiple OS's, but IE and Opera only on Windows
SportsTicker

- Provides real time sports information: scores, boxscores, news, etc
- Covers all popular US leagues in great detail. Less popular sports and foreign leagues get text based notes.
- Pushes XML data over a Kerberos authenticated socket
- 60 DTD's
- Volume: Thursday in July received 10,000 messages
- NFL stats came through two DTD's with a total of 268 fields
- Sunday in the real world: stats from the 14 NFL games found in 22 transmissions, 59,000 elements, 9,000 attributes
What is XML?

- Stands for Extensible Markup Language
- Similar to HTML: plain text using "<" and ">")" to delimit markup
- Markup components are named for the data they contain
- Became popular because it's a self explanatory way to transmit data
- More info: www.w3.org/XML/
XML Data: Sample and Explanation

```xml
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE NFLDSTAT SYSTEM "NFLDSTAT.dtd">
<NFLDSTAT>
  <DATE>2003-08-20</DATE>
  <SOURCE office="NY" />
  <PRECORD gender="M">
    <P_CODE>JOHNSONR</P_CODE>
    <P_PUNTYDS>53</P_PUNTYDS>
  </PRECORD>
  <PRECORD gender="F">
    <P_CODE>DIFRANCOA</P_CODE>
    <P_PUNTYDS>41</P_PUNTYDS>
  </PRECORD>
</NFLDSTAT>
```

- **XML Declaration**: specifies the version of XML being used
- **Document Type Declaration**: states the Root Element and DTD (Document Type Definition)
- **Elements**: are the building blocks of XML: start with "<" then element name
  - **Root Element**: the tags that hold the XML document ( `<NFLDSTAT>` )
  - **Elements**: (aka paired elements) have start and end tags, data can be in the start-tag's attributes and/or between the tags themselves ( `<DATE>` )
  - **Empty Elements**: single tag, closed by "/>", data in the attributes ( `<SOURCE.../>` )
- **Character Data**: information between elements ( `<JOHNSONR>` )
- **Attributes**: name/value pairs in start-tags ( `<gender="F"/>` )
General Parser Types

- Event Based
  - Goes through XML line by line
  - Read only
  - SAX (Simple API for XML)

- Tree Based
  - Places entire XML document into memory
  - Can move around, read and (often) write
  - DOM XML (Document Object Model)
  - SimpleXML
PHP's Other Parsers

- **DOM XML**
  - Experimental in PHP 4
  - Stable in PHP 5
  - But, your code will probably need to be modified
  - Documentation: [php.net/ref.domxml](http://php.net/ref.domxml)

- **XSLT**
  - API completely changed for PHP 5

- **Other Stable Extensions for PHP 5**
  - SimpleXML
  - XPath
  - Schema
PHP's SAX Parsers

- PHP 4 uses the Expat library: [www.jclark.com/xml/expat.html](http://www.jclark.com/xml/expat.html)

- PHP 5 uses the Libxml2 library: [www.xmlsoft.org/](http://www.xmlsoft.org/)

- Changes from PHP 4 to 5 are just in the back end. Don't need to modify your code.

- Documentation: [php.net/ref.xml](http://php.net/ref.xml)
SAX Parser Concepts

- Examines file linearly

- Each step type handled by a related user defined function
  - Start-tag
  - Character Data
  - End-tag
  - etc...

- Concepts for programming a parser
  - Produce the user defined functions
  - Obtain data
  - Declare the parser
  - Pass data to parser
function runParser() {
    /*
     * Replace all non-visible characters (except SP, TAB, LF and CR) with LF to keep the sax parser from choking.
     */
    $this->Contents = trim(preg_replace('/[^\x20-\x7E\x09\x0A\x0D]/', ' ', $this->Contents));
    $this->Contents = preg_replace('/&|&/i', '&', $this->Contents);

    $this->Parser = xml_parser_create('UTF-8');
    xml_set_object($this->Parser, $this);
    xml_set_element_handler($this->Parser, 'saxStartHandler', 'saxEndHandler');
    xml_set_character_data_handler($this->Parser, 'saxCharacterHandler');

    if (!xml_parse($this->Parser, $this->Contents, TRUE)) {
        $this->Probs[] = "File rejected by parser:
        " . xml_error_string(xml_get_error_code($this->Parser));
    }

    xml_parser_free($this->Parser);

    $ProbCount = count($this->Probs);
    if ($ProbCount != 0) {
        // Error handling omitted for clarity.
    }

    $this->IgnoreTheRest = 'N';
    $this->Contents = '';
    $this->ContentsRaw = '';
    $this->Data = array();
    $this->ParentElements = array();
    $this->Probs = array();

    return ($ProbCount == 0);
}
/**
 * Processes XML start tags.
 * Activated when an XML element opening tag is reached. The XML parser
 * automatically calls this function. Don't call this manually.
 *
 * @param   mixed    $Parser  variable to contain the current parser's reference id
 * @param   mixed    $Elem    variable to contain the current element's name
 * @param   mixed    $Attr    array to contain the current element's attributes
 */
function saxStartHandler(&$Parser, &$$Elem, &$$Attr) {
  if ($this->IgnoreTheRest == 'Y') {
    return;
  }

  array_push($this->ParentElements, $$Elem);

  // Is this a root element?
  if (count($this->ParentElements) == 1) {
    // If don't care about this file type, ignore the rest of it.
    if (!isset($this->RootElements['$$Elem'])) {
      $this->IgnoreTheRest = 'Y';
      return;
    }
  }

  foreach ($Attr AS $Key => $Value) {
    $this->Data['$$Elem:$Key'] = trim($Value);
  }

  $this->CData = array();
}
Parser: Character Data Handler

/**
 * Processes data between XML tags.
 *<p>Places character data from the current XML element
 * into a temporary array, $this->CData.  The XML parser
 * automatically calls this function each time a new line
 * of data is found between element tags.  Don't call this
 * manually.</p>
 *<p>We temporarily store the data because some
 * elements have multiple lines of information but the XML
 * parser only remembers the current line.</p>
 *<p>Don't call this function manually.</p>
 */

function saxCharacterHandler(&$Parser, & $Line) {
    if ($this->IgnoreTheRest == 'Y') {
        return;
    }
    $this->CData[] = $Line;
}
/**
 * Processes XML end tags.
 *<p>Determines what to do when we reach the end of each XML element. The XML parser automatically calls this function. Don't call this manually.</p>
 * @param mixed $Parser variable to contain the current parser's reference id
 * @param mixed $Elem variable to contain the current element's name
 */
function saxEndHandler(&$Parser, & $Elem) {
    if ($this->IgnoreTheRest == 'Y') {
        return;
    }
    $this->Data[$Elem] = trim(implode('', $this->CData));
    switch ($Elem) {
    case 'PRECORD':
        if (! $this->validateDataFields($Elem)) {
            break;
        }
        $this->runQuery($this->qsStatNflGamePlayerD($this->Data['P_CODE'], $this->Data['P_PUNTYDS']));
        $this->unsetFields($Elem);
        break;
    case 'LINESCORE':
        if (! $this->validateDataFields($Elem)) {
            break;
        }
        $this->runQuery($this->qsStatNflLinescore($this->Data['LINESCORE:TEAMCODE'], $this->Data['LINESCORE:QTR4']));
        $this->unsetFields($Elem);
    }
    array_pop($this->ParentElements);
}
Validation Required

- SportsTicker's data is unreliable
- Even if it was consistently good, insider could tamper with it
- Bad data can lead to failed queries or SQL injection attacks
Validation: Tables Define Types

**STP DataTypes**

<table>
<thead>
<tr>
<th>DataTypeID</th>
<th>RegularExpression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha10</td>
<td>/^[\w-]{1,10}$/</td>
</tr>
<tr>
<td>Int2</td>
<td>/^\d{1,2}$/</td>
</tr>
<tr>
<td>Int3Neg</td>
<td>/^-\d{1,2}</td>
</tr>
</tbody>
</table>

**STP DataFields**

<table>
<thead>
<tr>
<th>RootElement</th>
<th>ParentElement</th>
<th>DataField</th>
<th>DataTypeID</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFLDSTAT</td>
<td>PRECORD</td>
<td>P_CODE</td>
<td>Alpha10</td>
</tr>
<tr>
<td>NFLDSTAT</td>
<td>PRECORD</td>
<td>P_PUNYDS</td>
<td>Int3Neg</td>
</tr>
<tr>
<td>NFLBOXSCORE</td>
<td>LINESCORE</td>
<td>LINESCORE:QTR4</td>
<td>Int2</td>
</tr>
<tr>
<td>NFLBOXSCORE</td>
<td>LINESCORE</td>
<td>LINESCORE:TEAMCODE</td>
<td>Alpha10</td>
</tr>
</tbody>
</table>
/**
 * Establishes system settings.
 */
function getSettings() {
    // Data types.
    $this->DataTypes = array();
    $this->DataTypes =& $this->db->getAssoc( $this->qsDataTypeArray() );
    if ( DB::isError($this->DataTypes) ) {
        $this->killProcess('Having problems creating the DataTypes array.\');
    }
    // Data fields.
    $this->DataFields = array();
    $Result =& $this->db->query( $this->qsDataFieldArray() );
    if ( DB::isError($Result) ) {
        $this->killProcess('Having problems creating the DataFields array.\');
    }
    while ( $Result->fetchInto($Temp) ) {
        // Create a three dimensional array.
        $this->DataFields[$Temp['RootElement']][$Temp['ParentElement']][
            $Temp['DataField'] = $Temp['DataTypeID'];
    }
}
Validation: Resulting Arrays

$DataTypes
  [Alpha10] => /^\w-{1,10}$/
  [Int2] => /^\d{1,2}$/
  [Int3Neg] => /^-\d{1,2}|\d{1,3}$/

$DataFields
  [NFLDSTAT] => Array
    ( [PRECORD] => Array
      ( [P_CODE] => Alpha10
        [P_PUNTYDS] => Int3Neg
      )
    )
  [NFLBOXSCORE] => Array
    ( [LINESCORE] => Array
      ( [LINESCORE:QTR4] => Int2
        [LINESCORE:TEAMCODE] => Alpha10
      )
    )
/**
 * Validates data under the current element using the DataFields array.
 *
 * @param   string   $Elem  the current element name
 * @return  integer  1 if valid, 0 if not
 */
function validateDataFields($Elem) {
    // Ensure $DataFields array has validation types ready for this element.
    if ( empty($this->DataFields[$this->ParentElements[0]][$Elem])) {
        $this->Probs[] = "$Elem: DataFields[{$this->ParentElements[0]}][$Elem] is empty";
        $this->IgnoreTheRest = 'Y';
        return 0;
    }
    $Problems = 0;
    reset($this->DataFields[$this->ParentElements[0]][$Elem]);
    // > > GO THROUGH EACH FIELD NEEDED FROM THIS PARENT ELEMENT. < <
    foreach ($this->DataFields[$this->ParentElements[0]][$Elem] AS $Field => $Type) {
        // Ensure $DataTypes array has validation types ready for this type.
        if ( empty($this->DataTypes[$Type]) ) {
            $this->Probs[] = "$Elem: DataTypes[$Type] is empty";
            $Problems++;
            continue;
        }
        // If this field isn't set, don't even bother checking type.
        if ( !isset($this->Data[$Field]) ) {
            $this->Probs[] = "$Elem: $Field isn't set";
            $Problems++;
            continue;
        }
        // > > DOES THE DATA IN THIS FIELD MATCH THE EXPECTED TYPE? < <
        if ( !preg_match($this->DataTypes[$Type], $this->Data[$Field]) ) {
            $this->Probs[] = "$Elem: $Field does not match $Type: {$this->Data[$Field]}";
            $Problems++;
        }
    }
    if ( !empty($Problems) ) {
        $this->IgnoreTheRest = 'Y';
        return 0;
    }
    return 1;
}
More Info

- **PHP XML Parser Documentation**
  php.net/ref.xml

- **W3C Recommendation: Extensible Markup Language**
  www.w3.org/TR/REC-xml

- **The Analysis and Solutions Company**
  www.analysisandsolutions.com/

- **PHP XML Parsing Basics -- A Tutorial**
  www.analysisandsolutions.com/code/phpxml.htm

- **ST Parser**
  www.stparser.com/

- **SportsTicker**
  www.sportsticker.com/ticker/
/**
 * Processes the XML stream from STDIN.
 *<p>Reads STDIN. The data is accumulated in the class' $ContentsRaw variable. When the transmission separation character is reached this method executes the runParser() method.</p>
 *<p>If the connection to the WireParserClient is severed, even if intentionally by you, an error message will be generated that says "SportsTicker connection lost..."</p>
 */

function readStdinStream() {
    $In = fopen('php://stdin', 'r');
    while ( !feof($In) ) {
        $Line = fgets($In, 5000);
        $this->ContentsRaw .= $Line;
        if ( preg_match('/\x04/', $Line) ) {
            // Last line of transmission. Parse it.
            $this->runParser();
        }
    }
    $this->killProcess('SportsTicker connection lost: ' . date('Y-m-d H:i:s') );
}
** Updates the NFL team stats table with the linescore tag info
*
* @param string $TeamCode          the ST team id code
* @param integer $Qtr4              Points Team Scored During 4th Quarter
* @return string update query
*/
function qsStatNflLinescore($TeamCode, $Qtr4) {
    return "UPDATE STP_StatsNFLTeam SET Scored4Q=$Qtr4 " . "WHERE TeamCode=$TeamCode";
}

/**
* Updates NFL player stats with recovered dstat info
*
* @param string $PlayerCode        the ST player id code
* @param integer $P_PuntYds         Total Punting Yards
* @return string update query
*/
function qsStatNflGamePlayerD($PlayerCode, $P_PuntYds) {
    return "UPDATE STP_StatsNFLPlayer SET PuntYard=$P_PuntYds " . "WHERE PlayerCode=$PlayerCode";
/**
 * Executes the query string provided.
 * Errors from attempts to create records with duplicate keys are ignored.
 * Minor errors generate an email. Major errors shut down ST Parser.
 * @param string $Query the query string to execute
 */

function runQuery($Query) {

    $Result =& $this->db->query($Query);

    if ( DB::isError($Result) ) {
        switch ( $Result->getMessage() ) {
            case 'DB Error: already exists':
                // Generally means key duplicate. No problem.
                break;
            case 'DB Error: syntax error':
                break;
            case 'DB Error: invalid':
                break;
            case 'DB Error: invalid date or time':
                break;
            case 'DB Error: invalid number':
                $this->Probs[] = $Result->getMessage() . "\n" . $this->db->last_query;
                break;
            default:
                $this->killProcess($Result->getMessage() . "\n" . $this->db->last_query);
        }
    }
}
Appendix: Unsetting Old Data

/**
 * Unsets data under the current element using the DataFields array.
 * @param string $Elem the current element name
 */
function unsetFields($Elem) {
    foreach ($this->DataFields[$this->ParentElements[0]][$Elem] AS $Field => $Type) {
        unset($this->Data[$Field]);
    }
}
Appendix: $RootElements Array

**STP_RootElements**

<table>
<thead>
<tr>
<th>RootElement</th>
<th>Parse</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFLBOXSCORE</td>
<td>Y</td>
</tr>
<tr>
<td>NFLDSTAT</td>
<td>Y</td>
</tr>
</tbody>
</table>

```php
$RootElements
  [NFLDSTAT] =>
  [NFLBOXSCORE] =>
```
Appendix: $DataFields Array

**STP_DataFields**

<table>
<thead>
<tr>
<th>RootElement</th>
<th>ParentElement</th>
<th>DataField</th>
<th>DataTypeID</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFLDSTAT</td>
<td>PRECORD</td>
<td>P_CODE</td>
<td>Alpha10</td>
</tr>
<tr>
<td>NFLDSTAT</td>
<td>PRECORD</td>
<td>P_PUNTYDS</td>
<td>Int3Neg</td>
</tr>
<tr>
<td>NFLBOXSCORE</td>
<td>LINESCORE</td>
<td>LINESCORE:QTR4</td>
<td>Int2</td>
</tr>
<tr>
<td>NFLBOXSCORE</td>
<td>LINESCORE</td>
<td>LINESCORE:TEAMCODE</td>
<td>Alpha10</td>
</tr>
</tbody>
</table>

$DataFields

```php
[NFLDSTAT] => Array
    [
        [PRECORD] => Array
            [
                [P_CODE] => Alpha10
                [P_PUNTYDS] => Int3Neg
            ]
    ]

[NFLBOXSCORE] => Array
    [
        [LINESCORE] => Array
            [
                [LINESCORE:QTR4] => Int2
                [LINESCORE:TEAMCODE] => Alpha10
            ]
    ]
```
Appendix: $DataTypes Array

**STP_DataTypes**

<table>
<thead>
<tr>
<th>DataTypeID</th>
<th>RegularExpression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha10</td>
<td>^[\w-]{1,10}$</td>
</tr>
<tr>
<td>Int2</td>
<td>^\d{1,2}$/</td>
</tr>
<tr>
<td>Int3Neg</td>
<td>^-\d{1,2}</td>
</tr>
</tbody>
</table>

$DataTypes

[Alpha10] => ^[\w-]{1,10}$/
[Int2] => ^\d{1,2}$/
[Int3Neg] => ^-\d{1,2}|\d{1,3}$/
Table of Contents

- Table of Contents
- Title
- Introduction
- SportsTicker
- What is XML?
- XML Data: Sample and Explanation
- General Parser Types
- PHP's Other Parsers
- PHP's SAX Parsers
- SAX Parser Concepts
- Parser: Define and Execute
- Parser: Start Tag Handler
- Parser: Character Data Handler
- Parser: End Tag Handler
- Validation Required
- Validation: Tables Define Types
- Validation: Put Types Into Arrays
- Validation: Resulting Arrays
- Validation: Performing the Check
- More Info
- Appendix: Obtaining the XML
- Appendix: Query String Generation
- Appendix: Query Execution
- Appendix: Unsetting Old Data
- Appendix: $RootElements Array
- Appendix: $DataFields Array
- Appendix: $DataTypes Array