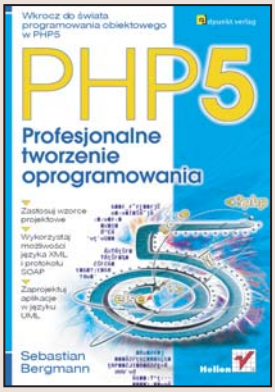
PHP5. Profesjonalne tworzenie oprogramowania



Autor: Sebastian Bergmann
Tłumaczenie: Wojciech Moch
ISBN: 83-246-0069-8
Tytuł oryginału: Professionelle Softwareentwicklung mit PHP 5
Format: B5, stron: 208

Kolejna wersja popularnego języka PHP wniosła zupełnie nową jakość do tworzenia aplikacji internetowych. PHP5 to w pełni obiektowe środowisko, pozwalające na korzystanie z wszystkich nowoczesnych technologii sieciowych i budowanie wydajnych oraz, co najważniejsze, bezpiecznych systemów. Przed twórcami aplikacji otwarły się bramy do protokołu SOAP, usług sieciowych, ogromnych możliwości języka XML i znacznie wydajniejszych połączeń z bazami danych.

Książka „PHP5. Profesjonalne tworzenie oprogramowania" jest przeznaczona właśnie dla takich programistów – tych, którzy opanowali poprzednie wersje PHP i chcą poznać możliwości, jakie oferuje jego najnowsze wcielenie. Przedstawia tajniki projektowania i programowania obiektowego, ze szczególnym uwzględnieniem stosowania wzorców projektowych i testowania za pomocą biblioteki PHPUnit. Opisuje możliwości zastosowania w aplikacjach PHP języka XML, protokołu SOAP i zaawansowanych technik operowania na bazach danych. Czytając ją, poznasz również metody automatycznego dokumentowania kodu oraz modelowania aplikacji za pomocą języka UML i dostępnych bezpłatnie narzędzi ArgoUML i Poseidon for UML.

- Klasy i obiekty
- Serializacja obiektów
- Mechanizmy dziedziczenia
- Korzystanie z biblioteki PHPUnit do testowania aplikacji
- Stosowanie wzorców projektowych
- Wzorce konstrukcyjne i strukturalne oraz wzorce zachowań
- Obsługa języka XML w PHP5
- Tworzenie usług sieciowych
- Wykorzystywanie możliwości rozszerzenia MySQL
- Komunikacja z bazami danych za pomocą Creole i Propel
- Tworzenie dokumentacji kodu z wykorzystaniem narzędzia phpDocumentator
- Modelowanie aplikacji w języku UML

Wykorzystaj najnowszą wersję PHP5
do stworzenia szybkich i bezpiecznych aplikacji internetowych

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 8.

Obsługa XML w PHP

„XML to gigantyczny krok naprzód,
w zupełnie żadnym kierunku”.

Eric Naggum

8.1. Wprowadzenie

W związku z cały czas rosnącą popularnością języka XML[1] (*eXtensible Markup Language*), w PHP5 wprowadzona została przygotowana od podstaw obsługa języka XML. W PHP4 standardowa instalacja pakietu była już wyposażona w sterowane zdarzeniami mechanizmy obsługi XML, bazujące na bibliotece *Expat*. Z kolei w standardowej instalacji PHP5 udostępnione zostały znacznie bardziej rozbudowane mechanizmy obsługi dokumentów XML, bazujące na bibliotekach *libxml2* i *libxslt* pochodzących z projektu GNOME.

- Rozszerzenie *SimpleXML* pozwala na bardzo prostą obsługę dokumentów XML.
- Rozszerzenie *XML* umożliwia obsługę dokumentów XML sterowaną zdarzeniami, zgodną z *Simple Api for XML* (SAX)[2], i w związku z tym jest zgodne z opartą na bibliotece *Expat* obsługą dokumentów XML, stosowaną w PHP4.
- Rozszerzenie *DOM* implementuje standard DOM (*Document Object Model*)[3], przez co pozwala na obiektową obsługę dokumentów XML reprezentowanych w postaci struktury drzewiastej.

---

1 *http://www.w3.org/XML/*

2 *http://www.saxproject.org/*

3 *http://www.w3.org/DOM/*

- Rozszerzenie *XSLT* udostępnia wszystkie funkcje konieczne do transformowania dokumentów zgodnie ze standardem XSL (*eXtensible Stylesheet Language*)[4].

8.2. SimpleXML

Do prostej obsługi dokumentów XML PHP udostępnia interfejs programistyczny *SimpleXML*. Interfejs ten opiera się na idei pakietu języka Perl o nazwie *XML::Simple* i w związku z tym pozwala na intuicyjny dostęp do danych zapisanych w dokumencie XML.

8.2.1. Ładowanie i zapisywanie dokumentów XML

Funkcje `simplexml_load_file()` i `simplexml_load_string()` tworzą obiekt klasy `SimpleXMLElement` przeznaczony do obsługi danych XML, które w zależności od użytej funkcji ładowane są ze wskazanego pliku lub ciągu znaków.

Metoda `asXML($filename)` zwraca dane istniejącego obiektu `SimpleXMLElement` w formacie XML i może je ewentualnie zapisać do wskazanego pliku.

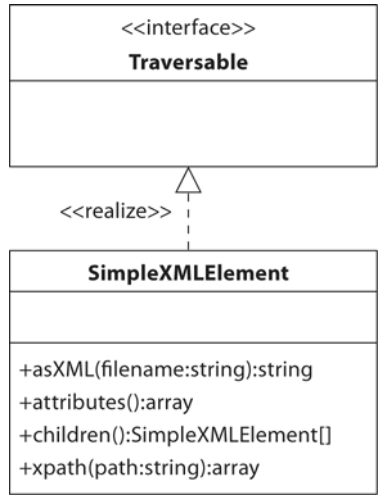
8.2.2. Klasa SimpleXMLElement

Obiekt klasy `SimpleXMLElement` (jej diagram UML przedstawiony został na rysunku 8.1) przechowuje po jednej zmiennej egzemplarza dla każdego podrzędnego elementu w strukturze XML. Każda taka zmienna otrzymuje nazwę opisującą dany element XML (proszę przyjrzeć się rysunkowi 8.2). Jeżeli dany element podrzędny występuje w strukturze tylko raz, to zmienna egzemplarza przechowuje wartość tego elementu. Jeżeli jednak dany element podrzędny występuje w strukturze wielokrotnie (tak jak element `<book>` na listingu 8.1), to zmienna egzemplarza przechowywać będzie tablicę[5] obiektów `SimpleXMLElement` opisujących każdy z elementów podrzędnych.

---

[4] *http://www.w3.org/Style/XSL/*

[5] Funkcje tablicowe PHP nie będą jednak współpracowały z taką tablicą, a wywołanie `is_array($books->book[1]->author)` zwróci wartość `FALSE`. Mimo to zawarte w PHP funkcje debugowania `print_r()` i `var_dump()` będą traktować zapis `$books->book[1]->author` jako tablicę.



Rysunek 8.1. *Klasa SimpleXMLElement*

```
SimpleXMLElement Object
(
    [book] => Array
        (
            [0] => SimpleXMLElement Object
                (
                    [author] => Sebastian Bergmann
                    [title] => Profesjonalne tworzenie oprogramowania w PHP 5
                    [isbn] => 83-246-0069-8
                )

            [1] => SimpleXMLElement Object
                (
                    [author] => Array
                        (
                            [0] => Emilian Balanescu
                            [1] => Mihai Bucica
                            [2] => Cristian Darie
                        )

                    [title] => PHP5 i MySQL. Zastosowania e-commerce
                    [isbn] => 83-7361-830-9
                )
        )
)
```

Rysunek 8.2. *Reprezentacja plików XML w obiektach typu SimpleXMLElement*

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<!DOCTYPE books PUBLIC "books" "books.dtd">

<books>
  <book lang="pl">
    <author>Sebastian Bergmann</author>
    <title>Profesjonalne tworzenie oprogramowania w PHP 5</title>
    <isbn>83-246-0069-8</isbn>
  </book>
```

Listing 8.1. *Katalog książek w formacie XML*

```
  <book lang="de">
    <author>Emilian Balanescu</author>
    <author>Mihai Bucica</author>
    <author>Cristian Darie</author>
    <title>PHP5 i MySQL. Zastosowania e-commerce</title>
    <isbn>83-7361-830-9</isbn>
  </book>
</books>
```

Obiekt klasy `SimpleXMLElement` wyposażony jest w implementację metody przechwytującej `__toString` (była o niej mowa w podrozdziale 2.6), dlatego możliwe jest odczytanie tekstowej reprezentacji każdego elementu XML (`CDATA`), na przykład za pośrednictwem instrukcji `print`. Na listingu 8.2 przedstawiony został sposób odczytywania ze struktury XML zawartości elementu `title` z pierwszego elementu `book`.

Listing 8.2. *Dostęp do elementu XML za pomocą interfejsu SimpleXML*

```
<?php
$books = simplexml_load_file('books.xml');

print $books->book[0]->title;
?>
```

```
Profesjonalne tworzenie oprogramowania w PHP 5
```

W obiektach `SimpleXMLElement` metoda `attributes()` zwraca tablicę asocjacyjną ze wszystkimi atrybutami danego elementu XML. Przykład wykorzystania tej metody przedstawiony został na listingu 8.3.

Listing 8.3. *Dostęp do atrybutów elementu XML poprzez interfejs SimpleXML*

```
<?php
$books = simplexml_load_file('books.xml');

foreach ($books->book[0]->attributes() as $nazwa=>$wartosc) {
  print "$nazwa: $wartosc\n";
}
?>
```

```
lang: pl
```

Klasa `SimpleXMLElement` implementuje również interfejs iteratora `Traversable` (omawiany był w rozdziale 3.). Dzięki temu obiekt typu `SimpleXMLElement` może być dowolnie iterowany za pomocą operatora `foreach` (i jednocześnie można obsługiwać zawarte w nim dane dokumentu XML), tak jak na listingu 8.4.

Listing 8.4. *Dostęp do katalogu książek za pomocą interfejsu SimpleXML i operatora foreach*

```
<?php
$books = simplexml_load_file('books.xml');

foreach ($books->book as $ksiazka) {
  $autorzy = '';

  foreach ($ksiazka->author as $autor) {
    $autorzy .= empty($autorzy) ? $autor : ', ' . $autor;
  }

  printf(
    "%s: \"%s\"\nISBN: %s\n\n",
```

```
    $autorzy,
    $ksiazka->title,
    $ksiazka->isbn
  );
}
?>
```

```
Sebastian Bergmann: "Profesjonalne tworzenie oprogramowania w PHP 5"
ISBN: 83-246-0069-8
Emilian Balanescu, Mihai Bucica, Cristian Darie: "PHP5 i MySQL. Zastosowania
e-commerce"
ISBN: 83-7361-830-9
```

Zawartość obiektów `SimpleXMLElement` może zostać zmieniona przez odpowiednie modyfikacje odpowiednich zmiennych egzemplarza, tak jak na listingu 8.5.

Listing 8.5. *Zmiana zawartości obiektu SimpleXMLElement i wypisanie danych XML*

```
<?php
$books = simplexml_load_file('books.xml');
$books->book[0]->author = 'Johannes Sebastian Bergmann';

print $books->book[0]->asXML();
?>
```

```
<book lang="pl">
    <author>Johannes Sebastian Bergmann</author>
    <title>Profesjonalne tworzenie oprogramowania w PHP 5</title>
    <isbn>83-246-0069-8</isbn>
  </book>
```

XPath

Korzystając w obiekcie `SimpleXMLElement` z metody `xpath()`, można tworzyć zapytania w języku XML Path Language (XPath)[6]. Na listingu 8.6 zostało przedstawione zapytanie zwracające tytuły wszystkich książek zapisanych w katalogu. Wynikiem takich zapytań jest zawsze tablica obiektów typu `SimpleXMLElement`.

Listing 8.6. *Zastosowanie języka XPath w interfejsie SimpleXML*

```
<?php
$books = simplexml_load_file('books.xml');

foreach ($books->xpath('book/title') as $tytul) {
  print "$tytul\n";
}
?>
```

```
Profesjonalne tworzenie oprogramowania w PHP 5
PHP 5
```

8.2.3. Klasa SimpleXMLIterator

W bibliotece standardowej PHP (była o niej mowa w rozdziale 3.) dostępna jest też pewna alternatywa dla klasy `SimpleXMLElement`, pozwalająca na obsługę dokumentów XML za pomocą interfejsu *SimpleXML* — klasa `SimpleXMLIterator`.

[6] *http://www.w3.org/TR/xpath*

Klasa `SimpleXMLIterator` udostępnia interfejs `RecursiveIterator` i dlatego może przeglądać wszystkie elementy dokumentu XML, korzystając przy tym z dodatkowego iteratora typu `RecursiveIteratorIterator`.

Na listingu 8.7 wywoływana jest funkcja `simplexml_load_file()`, której podawany jest opcjonalny drugi parametr, przez co w wyniku swojego działania funkcja ta, zamiast obiektu `SimpleXMLElement`, zwraca obiekt typu `SimpleXMLIterator`. Następnie tworzony jest obiekt typu `RecursiveIteratorIterator` obsługujący obiekt `SimpleXMLIterator`. Przy użyciu tego iteratora, w wyniku podania dodatkowego parametru `RIT_SELF_FIRST`, wypisywane są wszystkie elementy XML z elementem podstawowym (root) włącznie.

Listing 8.7. *Rekursywne wyliczanie wszystkich elementów dokumentu XML*

```
<?php
$dokument = simplexml_load_file(
  'books.xml',
  'SimpleXMLIterator'
);

$iterator = new RecursiveIteratorIterator(
  $dokument,
  RIT_SELF_FIRST
);

foreach($iterator as $nazwa => $element) {
  print $nazwa . "\n";
}
?>
```

```
book
author
title
isbn
book
author
author
author
title
isbn
```

Wewnątrz pętli `foreach` z listingu 8.7 zmienna `$nazwa` przechowuje nazwę aktualnego elementu XML, natomiast w zmiennej `$element` zapisywany jest obiekt opisujący ten element.

8.2.4. Wady i zalety

- Zaleta: Bardzo prosty dostęp do elementów XML.
- Zaleta: Niewielkie zapotrzebowanie na pamięć, ponieważ obiekty PHP tworzone są tylko w razie potrzeby, a sam dokument początkowo zapisany jest tylko raz (w pamięci biblioteki *libxml2*).
- Zaleta: Prędkość działania.
- Wada: Utrudniony dostęp do atrybutów elementów XML.
- Wada: Brak standardu.

8.3. Simple API for XML (SAX)

Orientacja na zdarzenia

W czasie obsługi danych XML za pomocą Simple API for XML (SAX) parser XML wywołuje określone zdarzenia (otwierający znacznik elementu, zamykający znacznik elementu itp.), które muszą być obsłużone przez programistę w ramach specjalnych metod lub funkcji.

Najpierw przyjrzymy się trzem najważniejszym zdarzeniom wykorzystywanym w trakcie przetwarzania danych XML: „Start Element", „End Element" i „Character Data".

- `xml_set_element_handler($parser, <start_element_handler>, <end_element_handler>)` — Rejestruje funkcje wywołań zwrotnych dla zdarzeń „Start Element" i „End Element". Funkcja lub metoda obsługująca zdarzenie „Start Element" musi przyjmować trzy parametry: `$parser`, `$name` i `$attributes`. Ostatni z tych parametrów zawiera tablicę asocjacyjną z atrybutami elementu XML. Funkcja lub metoda obsługująca zdarzenie „End Element" musi przyjmować dwa parametry: `$parser` i `$name`.
- `xml_set_character_data_handler($parser, <handler>)` — Rejestruje funkcję wywołań zwrotnych dla zdarzenia „Character Data". Funkcja lub metoda, która ma obsługiwać to zdarzenie, musi przyjmować dwa parametry: `$paser` i `$cdata`.

W czasie rejestrowania funkcji wywołania zwrotnego (ang. *callback*) w parametrze `<handler>` można przekazać nazwę funkcji lub tablicy przechowującej referencję obiektu, a także nazwę metody tego obiektu, która ma obsługiwać dane zdarzenie.

Na listingu 8.8 przedstawiony został sposób obsługi katalogu książek z listingu 8.1.

Listing 8.8. *Dostęp do katalogu książek poprzez interfejs SAX*

```
<?php
class Ksiazki {
  protected $parser;
  protected $stosElementow = array();
  protected $autorzy = array();

  public function __construct() {
    $this->parser = xml_parser_create();

    xml_set_object($this->parser, $this);

    xml_set_element_handler(
      $this->parser,
      'startElement',
      'endElement'
    );

    xml_set_character_data_handler(
      $this->parser,
      'characterData'
    );
```

```
    xml_set_default_handler(
      $this->parser,
      'characterData'
    );
  }

  public function __destruct() {
    xml_parser_free($this->parser);
  }

  public function odczytajKatalog($katalog) {
    xml_parse(
      $this->parser,
      file_get_contents($katalog),
      TRUE
    );
  }

  protected function
  startElement($parser, $element, $attributes) {
    array_push($this->stosElementow, $element);

    if ($element == 'BOOK') {
      $this->autorzy = array();
    }
  }

  protected function endElement($parser, $element) {
    array_pop($this->stosElementow);
  }

  protected function characterData($parser, $cdata) {
    $poziom   = sizeof($this->stosElementow) - 1;
    $element = $this->stosElementow[$poziom];

    switch($element) {
        case 'AUTHOR': {
          $this->autorzy[] = $cdata;
        }
        break;

        case 'TITLE': {
          print implode(', ', $this->autorzy) . "\n";
          print $cdata . "\n";
        }
        break;

        case 'ISBN': {
          print 'ISBN: ' . $cdata . "\n\n";
        }
        break;
    }
  }
}
```

```
$ksiazki = new Ksiazki;
$ksiazki->odczytajKatalog('books.xml');
?>
```

```
Sebastian Bergmann
Profesjonalne tworzenie oprogramowania w PHP5
ISBN: 83-246-0069-8

Emilian Balanescu, Mihai Bucica, Cristian Darie
PHP5 i MySQL. Zastosowania e-commerce
ISBN: 83-7361-830-9
```

Parser interfejsu SAX przetwarza dane dokumentu XML w sposób liniowy i w pamięci przechowuje tylko te dane, które związane są z bieżącym zdarzeniem. Dlatego też programista musi się zajmować przechowywaniem danych aktualnie obsługiwanego elementu XML. W naszym przykładzie nazwę otwieranego właśnie znacznika XML zapisujemy na stosie. W metodzie obsługującej zdarzenie „Character Data" można dzięki temu stwierdzić, z którym elementem związane są aktualnie obsługiwane dane. W momencie gdy parser dotrze do zamykającego znacznika XML, nazwę elementu można usunąć ze stosu.

W interfejsie SAX obok zdarzeń „Start Element", „End Element" i „Character Data" udostępniane są jeszcze następujące zdarzenia:

- `xml_set_processing_instruction_handler($parser, <handler>)` — rejestruje funkcję wywołania zwrotnego obsługującą zdarzenie „Processing Instruction".
- `xml_set_external_entity_ref_handler($parser, <handler>)` — rejestruje funkcję wywołania zwrotnego obsługującą zdarzenie „External Entity Reference".
- `xml_set_notation_decl_handler($parser, <handler>)` — rejestruje funkcję wywołania zwrotnego obsługującą zdarzenie „Notation Declaration".
- `xml_set_unparsed_entity_decl_handler($parser, <handler>)` — rejestruje funkcję wywołania zwrotnego obsługującą zdarzenie „Unparsed Entity Declaration".
- `xml_set_default_handler($parser, <handler>)` — rejestruje funkcję wywołania zwrotnego obsługującą wszystkie te zdarzenia, dla których nie została zarejestrowana metoda obsługi.

8.3.1. Wady i zalety

- Zaleta: Standard, choć niezatwierdzony przez konsorcjum WWW.
- Zaleta: Niewielkie użycie pamięci, ponieważ w danym momencie dostępny jest tylko aktualny element XML.
- Zaleta: Szybkość działania.

- Wada: Praca liniowa, nie jest możliwy swobodny dostęp do elementów.
- Wada: Nie ma możliwości zapisu danych.
- Wada: Programista musi zajmować się wszystkimi szczegółami obsługi.

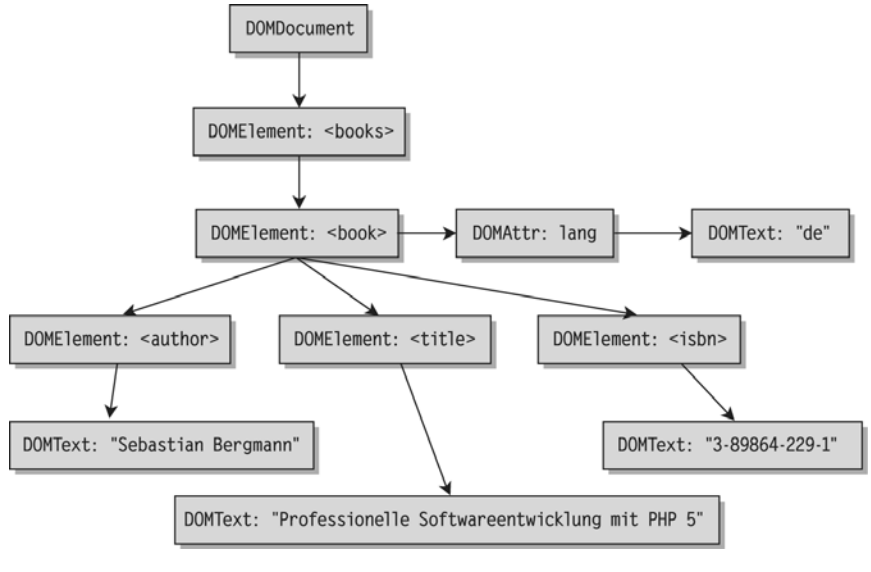
8.4. Document Object Model (DOM)

Konsorcjum WWW (W3C)[7] przygotowało bardzo potężny standard w postaci modelu DOM (*Document Object Model*), stanowiącego niezależny od platformy i języka programowania interfejs programistyczny przeznaczony do odczytywania i zapisywania danych w prawidłowych dokumentach HTML i prawidłowo zbudowanych dokumentach XML.

PHP5 udostępnia własne rozszerzenie obsługi modelu DOM, które niestety nie jest zgodne z rozszerzeniem obsługi modelu DOM z PHP4. Rozszerzenie to jest pełną implementacją standardu DOM w wersji „Level 2 Core", a dodatkowo zawiera pewne elementy („Load and Save") nowszej wersji „Level 3".

Węzły

Model DOM zajmuje się wszystkimi elementami dokumentu XML, które traktowane są jak węzły drzewa. Sam dokument (`DOMDocument`), jego elementy (`DOMElements`), a także ich atrybuty (`DOMAttr`) oraz tekstowa zawartość (`DOMText`) reprezentowane są przez obiekty odpowiednich klas węzłów, które powiązane są ze sobą za pomocą referencji. Na rysunku 8.3 przedstawione zostało drzewo obiektów DOM utworzone na podstawie dokumentu XML prezentowanego na listingu 8.1.



Rysunek 8.3. *Reprezentacja struktury katalogu książek w postaci struktury DOM*

[7] *http://www.w3.org/*

Implementacja modelu DOM w PHP5 obejmuje 29 klas w sumie zawierających 360 metod. W tym miejscu zajmiemy się najważniejszymi z nich (przedstawione zostały w tabeli 8.1).

Tabela 8.1. *Najważniejsze klasy modelu DOM*

Klasa	Klasa bazowa lub interfejs	Zadanie
`DOMNode`		Klasa bazowa dla pozostałych klas węzłów drzewa DOM.
`DOMDocument`	`DOMNode`	Stanowi punkt wejścia do drzewa DOM.
`DOMElement`	`DOMNode`	Reprezentuje element XML i udostępnia metody dostępu do elementów XML.
`DOMAttr`	`DOMNode`	Reprezentuje atrybut elementu XML.
`DOMCharacterData`	`DOMNode`	Reprezentuje część danych tekstowych elementu XML.
`DOMText`	`DOMCharacterData`	Reprezentuje tekstowe zawartości dokumentu XML.
`DOMNodeList`	`Traversable`	Reprezentuje zbiór węzłów.
`DOMXPath`		Umożliwia formułowanie zapytań XPath.
`DOMException`	`Exception`	Klasa wyjątku.

8.4.1. Ładowanie i zapisywanie dokumentów XML

Metoda `DOMDocument::load($filename)` przeznaczona jest do odczytywania dokumentu XML ze wskazanego pliku i może być wywoływana zarówno na rzecz konkretnego obiektu, jak i statycznie na rzecz klasy. Jeżeli wywoływana jest statycznie, to zwraca w wyniku swojego działania obiekt klasy `DOMDocument` (przykład takiego działania znaleźć można na listingu 8.12). Z kolei metoda `DOMDocument::loadXML($xml)` pozwala na załadowanie dokumentu XML z ciągu znaków, natomiast funkcja `dom_import_simplexml($sxe)` tworzy nowy obiekt klasy `DOMDocument` na podstawie obiektu klasy `SimpleXMLElement`.

Podobnie metody `DOMDocument::loadHTMLFile($filename)` i `DOMDocument::loadHTML($html)` pozwalają na załadowanie dokumentu HTML z pliku lub ciągu znaków.

Metody `DOMDocument::save($filename)` i `DOMDocument::saveXML()` przeznaczone są do zapisywania dokumentu XML we wskazanym pliku lub ciągu znaków. Do obsługi zapisu dokumentów HTML przygotowane zostały metody `DOMDocument::saveHTMLFile($filename)` i `DOMDocument::saveHTML()`.

Strumienie

Wszystkie wymienione metody wykorzystują system strumieni funkcjonujący w PHP5. Tym samym w zależności od tego, która wersja obsługi strumieni zostanie wkompilowana do projektu, metody te mogą na przykład otworzyć plik XML

ze zdalnego serwera albo zapisać dokument XML na serwerze FTP. W takich przypadkach zamiast nazwy pliku w parametrze `$filename` należy podać odpowiedni adres URL.

8.4.2. Tworzenie nowego dokumentu XML

Na listingu 8.9 najpierw tworzony jest obiekt klasy `DOMDocument`. Konstruktorowi tej klasy przekazywana jest żądana wersja standardu XML (`"1.0"`) oraz stosowane w dokumencie kodowanie znaków (`"iso-8859-2"`).

Listing 8.9. *Tworzenie nowego dokumentu XML*

```
<?php
$dokument = new DOMDocument('1.0', 'iso-8859-2');
$dokument->formatOutput = TRUE;

$ksiazki = $dokument->appendChild(
  $dokument->createElement('books')
);

$ptop5 = $ksiazki->appendChild(
  $dokument->createElement('book')
);

$ptop5->setAttribute('lang', 'pl');

$ptop5->appendChild(
  $dokument->createElement(
    'author',
    'Sebastian Bergmann'
  )
);

$ptop5->appendChild(
  $document->createElement(
    'title',
    'Profesjonalne tworzenie oprogramowania w PHP 5'
  )
);

$ptop5->appendChild(
  $dokument->createElement(
    'isbn',
    '83-246-0069-8'
  )
);

print $dokument->saveXML();
?>
```

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<books>
  <book lang="pl">
    <author>Sebastian Bergmann</author>
    <title>Profesjonalne tworzenie oprogramowania w PHP 5</title>
    <isbn>83-246-0069-8</isbn>
  </book>
</books>
```

Metoda `DOMDocument::createElement()` tworzy nowy obiekt klasy `DOMElement`, z kolei za pomocą metody `DOMNode::appendChild()` można „podwiesić" utworzony element w odpowiednim miejscu w drzewie DOM.

Tworzenie nowego dokumentu XML za pomocą klas modelu DOM, tak jak przedstawione to zostało na listingu 8.9, może okazać się zadaniem wyjątkowo pracochłonnym. W takich przypadkach dobrze jest przygotować zestaw klas rozbudowujących klasy modelu DOM poprzez dziedziczenie i dopasować je do potrzeb aktualnej aplikacji.

Klasa `Book` (prezentowana na listingu 8.10) rozbudowuje klasę `DOMElement` o metody `setAuthor($author)`, `setISBN($isbn)`, `setLanguage($language)` i `setTitle ($title)`, ściśle powiązane z elementem `<book>` w drzewie dokumentu XML. W ten sposób użytkownik tej klasy nie musi już samodzielnie wywoływać metod `DOMDocument::createElement()` i `DOMNode::appendChild()`.

Listing 8.10. *Klasa Book*

```
<?php
class Book extends DOMElement {
  private $document;

  public function __construct(DOMDocument $document) {
    parent::__construct('book');
    $this->document = $document;
  }

  public function setAuthor($author) {
    $this->appendChild(
      $this->document->createElement(
        'author',
        $author
      )
    );
  }

  public function setISBN($isbn) {
    $this->appendChild(
      $this->document->createElement(
        'isbn',
        $isbn
      )
    );
  }

  public function setLanguage($language) {
    $this->setAttribute('lang', $language);
  }

  public function setTitle($title) {
    $this->appendChild(
      $this->document->createElement(
        'title',
        $title
      )
    );
  }
}
?>
```

Klasa `Books` (prezentowana na listingu 8.11) rozbudowuje klasę `DOMDocument`, dodając do niej metodę `createBook()` pozwalającą w prosty sposób utworzyć element typu `<book>`.

Listing 8.11. *Klasa Books*

```
<?php
require_once 'Book.php';

class Books extends DOMDocument {
  private $books;

  public function __construct() {
    parent::__construct('1.0', 'iso-8859-2');

    $this->books = $this->appendChild(
      $this->createElement('books')
    );

    $this->formatOutput = TRUE;
  }

  public function createBook() {
    $book = new Book($this);
    $this->books->appendChild($book);

    return $book;
  }
}

$books = new Books;
$book  = $books->createBook();

$book->setAuthor('Sebastian Bergmann');
$book->setISBN('83-246-0069-8');
$book->setTitle(
  'Profesjonalne tworzenie oprogramowania w PHP 5'
);
$book->setLanguage('pl');

print $books->saveXML();
?>
```

Jak można się domyślić, kod z listingu 8.11 tworzy dokładnie taki sam dokument jak kod z listingu 8.9.

8.4.3. Wykorzystywanie klasy DOMNodeList

Klasa `DOMNodeList` reprezentuje zbiór węzłów, czyli obiektów klasy `DOMNode`. Węzły w takim zbiorze nie są układane według żadnej konkretnej reguły. W klasie `DOMNodeList` implementowany jest interfejs `Traversable` (była o nim mowa w rozdziale 3.), dlatego możliwe jest iterowanie przez poszczególne obiekty listy z wykorzystaniem operatora `foreach`.

Na listingu 8.12 korzystamy z metody `DOMDocument::getElementsByTagName()`, aby w ten sposób uzyskać listę wszystkich elementów typu `<author>`. W wyniku swojego działania metoda ta zwraca obiekt klasy `DOMNodeList`, w którym zawarte są odpowiednie obiekty typu `DOMElement`. Następnie za pomocą zapisu `$author->nodeValue` odczytywane są wartości węzłów typu `DOMText`, które dołączone są do obiektów typu `DOMElement` zawartych w zmiennej `$autor`.

Listing 8.12. *Zastosowanie klasy DOMNodeList*

```
<?php
$dokument = DOMDocument::load('books.xml');
$autorzy  = $dokument->getElementsByTagName('author');

foreach ($autorzy as $autor) {
  print $autor->nodeValue . "\n";
}
?>
```

```
Sebastian Bergmann
Emilian Balanescu
Mihai Bucica
Cristian Darie
```

8.4.4. Formułowanie zapytań XPath

XPath

Za pomocą klasy `DOMXPath` można tworzyć zapytania *XPath* dotyczące drzewa DOM. Wynikiem takiego zapytania zawsze jest obiekt klasy `DOMNodeList`. Przykład zastosowania tej klasy przedstawiony został na listingu 8.13.

Listing 8.13. *Formułowanie zapytań XPath*

```
<?php
$dokument = DOMDocument::load('books.xml');
$xpath    = new DOMXPath($dokument);

foreach ($xpath->query('book/title') as $tytul) {
  print $tytul->nodeValue . "\n";
}
?>
```

```
Profesjonalne tworzenie oprogramowania w PHP 5
PHP5 i MySQL. Zastosowania e-commerce
```

8.4.5. Sprawdzanie poprawności dokumentów XML

Rozszerzenie PHP-DOM obok obsługi i tworzenia dokumentów XML pozwala kontrolować zgodność dokumentów XML ze specyfikacjami zapisanymi w formatach DTD (*Document Type Definition*), XSD (*XML Schema*) lub RNG (*RelaxNG*).

XML Schema

XML Schema[8] jest oficjalnie zatwierdzoną przez W3C technologią zastępującą format Document Type Definition. Tworzone schematy stosowane są do definiowania klas dokumentów XML. Na podstawie przygotowanego schematu można skontrolować, czy konkretny dokument XML spełnia założenia takiego schematu. Taki proces nazywany jest sprawdzaniem zgodności (ang. validate) dokumentu XML ze schematem.

Na listingu 8.14 zastosowana została metoda `DOMDocument::schemaValidate()` udostępniana przez PHP5. W tym prostym programie kontrolowana jest zgodność dokumentu *books.xml* ze schematem XML zapisanym w pliku *books.xsd* (zawartość tego pliku prezentowana jest na listingu 8.15).

Listing 8.14. *Kontrola zgodności ze schematem XML*

```
<?php
$dokument = DOMDocument::load('books.xml');

if ($dokument->schemaValidate('books.xsd')) {
  print 'Plik books.xml jest zgodny z deklaracją schematu '.
        'zapisaną w pliku books.xsd.';
}
?>
```

```
Plik books.xml jest zgodny z deklaracją schematu zapisaną w pliku books.xsd.
```

Listing 8.15. *Plik schematu XML przygotowany dla dokumentu katalogu książek*

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="books">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" ref="book"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="author" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="title"  minOccurs="1"/>
        <xsd:element ref="isbn"   minOccurs="1"/>
      </xsd:sequence>

      <xsd:attribute name="lang" use="required" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="author" type="xsd:string"/>
  <xsd:element name="title"  type="xsd:string"/>
  <xsd:element name="isbn"   type="xsd:string"/>
</xsd:schema>
```

[8] *http://www.w3.org/XML/Schema*

RelaxNG

RelaxNG jest językiem schematów XML przeznaczonym do definiowania struktury dokumentów XML, funkcjonującym równolegle z językiem XML Schema. Język ten jest obecnie standaryzowany przez organizację ISO, ale nie jest wspierany przez konsorcjum W3C, choć wykorzystuje typy danych pochodzące z języka XML Schema. Najprawdopodobniej już wkrótce język RelaxNG stanie się alternatywą dla języka XML Schema, znacznie prostszą do poznania.

Na listingu 8.16 przedstawiony został sposób wykorzystania metody `DOMDocument::relaxngValidate()`. W tym prostym programie kontrolowana jest zgodność dokumentu *books.xml* z definicją RelaxNG zapisaną w pliku *books.rng* (zawartość tego pliku prezentowana jest na listingu 8.17).

Listing 8.16. *Kontrola zgodności z definicją RelaxNG*

```
<?php
$dokument = DOMDocument::load('books.xml');

if ($dokument->relaxngValidate('books.rng')) {
  print 'Plik books.xml jest zgodny z deklaracją RelaxNG '.
        'zapisaną w pliku books.rng.';
}
?>
```

```
Plik books.xml jest zgodny z deklaracją RelaxNG zapisaną w pliku books.rng.
```

Listing 8.17. *Plik deklaracji RelaxNG przygotowany dla dokumentu katalogu książek*

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

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <start>
    <element name="books">
      <oneOrMore>
        <element name="book">
          <attribute name="lang">
            <data type="NMTOKEN"/>
          </attribute>

          <oneOrMore>
            <element name="author">
              <text/>
            </element>
          </oneOrMore>

          <element name="title">
            <text/>
          </element>

          <element name="isbn">
            <text/>
          </element>
        </element>
      </oneOrMore>
    </element>
  </start>
</grammar>
```

8.4.6. Wady i zalety

- Zaleta: Implementacja potężnego standardu.
- Zaleta: Możliwość odczytywania i zapisywania danych dowolnego elementu XML.
- Wada: Powolne działanie i duże zapotrzebowanie na pamięć, ponieważ tworzone jest pełne drzewo obiektów opisujące dokument XML.

8.5. Transformacje XSL (XSLT)

Na obecnym etapie rozwoju aplikacji WWW nie można w nich pominąć operacji przekształcania danych z formatu XML na inny format, na przykład HTML lub PDF. Do realizacji tego zadania konsorcjum W3C proponuje nam język XSL9 (*eXtensible Stylesheet Language*). Język ten składa się z trzech komponentów:

- Język *XSL Transformations* (XSLT)10 stanowi osobny język programowania opisujący transformacje XML.
- Język *XML Path Language* (XPath) wykorzystywany jest w języku XSLT do uzyskiwania dostępu do całości dokumentu XML lub jego części.
- Język *XSL Formatting Objects* (XSL-FO) stosowany jest do opisywania formatowania danych.

Pod względem zasady działania język XSLT jest spokrewniony z uniksowym narzędziem *awk*. Warunki (*Template Match*) albo funkcje (*Template Name*) stosowane są po kolei na danych wejściowych. Dla narzędzia *awk* dane te składają się z wierszy tekstu, natomiast dla języka XSLT danymi wejściowymi jest struktura drzewiasta. Język XSLT, podobnie jak *awk*, nie może korzystać z przygotowanych przez siebie danych wyjściowych.

Programy napisane w języku XSLT właściwie są tylko dokumentami XML, a ze względów historycznych nazywane są arkuszami stylów. Z tego właśnie powodu pisanie w tym języku jest wyjątkowo czasochłonne, ponieważ nawet najprostsze konstrukcje wymagają wpisywania dużej ilości tekstu. Na listingu 8.18 przedstawiony został kod wymagany do zrealizowania w języku XSLT prostej konstrukcji `if-then-else`, natomiast na listingu 8.19 — odpowiadający mu wycinek kodu w języku PHP.

Listing 8.18. *Konstrukcja if-then-else w języku XSLT*

```
<xsl:template match="/">
  <xsl:choose>
    <xsl:when test="$foobar = 'foo'">
      <xsl:call-template name="foo"/>
```

[9] *http://www.w3.org/Style/XSL/*

[10] *http://www.w3.org/TR/xslt*

```
      </xsl:when>
      <xsl:otherwise>
        <xsl:call-template name="bar"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

```
<?php
if ($foobar == 'foo') foo() else bar();
?>
```

Listing 8.19. *Konstrukcja if-then-else w języku PHP*

Język transformacji

Podobnie jak język DSSSL (*Document Style Semantics and Specification Language*)[11] dla języka SGML, język XSLT jest funkcyjnym językiem transformacji dla języka XML. Oznacza to, że zawiera on wszystkie wady języków funkcjonalnych, takie jak wykorzystywanie rekursji do tworzenia pętli albo obsługa zmiennych, które są bardziej stałymi niż zmiennymi. Poza tym nie pozwala korzystać z zalet programowania funkcjonalnego, czyli programowania z domknięciami albo obliczeń korzystających z funkcji jako typów danych. Takich możliwości język XSLT nie udostępnia.

Kolejną wadą języka XSLT jest ograniczenie jego domeny. Co prawda „wie on wszystko" na temat XML i udostępnia wiele funkcji obsługujących takie dokumenty, ale niestety nie ma wielkich możliwości działania w dziedzinach wykraczających poza XML. Bez wykorzystywania innych języków programowania, takich jak Java, dołączanych do programów tworzonych w języku XSLT poprzez funkcje wywołań zwrotnych, programy te nie są w stanie wykonać jakiejkolwiek operacji poza światem XML. Bez zewnętrznej pomocy nie jest w nich możliwe dołączenie danych z innych źródeł, na przykład baz danych, albo generowanie danych w formatach nietekstowych, na przykład grafik lub plików PDF.

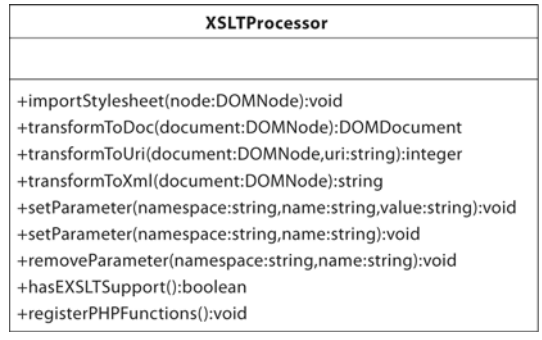
8.5.1. Interfejs programowy XSLT w PHP

PHP5 udostępnia podobną do interfejsu programistycznego projektu *Mozilla* klasę `XSLTProcessor` (prezentowaną na rysunku 8.4), służącą do wykonywania transformacji XSLT. W przeciwieństwie do omawianych wcześniej interfejsów programowej obsługi formatu XML, rozszerzenie obsługi XSLT nie jest dołączane do standardowej konfiguracji PHP5. Musi być ono specjalnie aktywowane w czasie budowania pakietu poprzez dodanie w wierszu poleceń parametru `--with-xsl` (proszę przejrzeć Dodatek A).

Na listingu 8.21 za pomocą metody `DOMDocument::load()` tworzony jest obiekt klasy `DOMDocument`, do którego ładowane są zarówno dokument XML (przedstawiony na listingu 8.1), jak i arkusz stylów XSL (prezentowany na listingu 8.20). Poprzez metody `importStylesheet()` i `transformToDoc()` obiekt arkusza stylów XLS przekazywany jest do obiektu procesora XSLT, a dokument wejściowy transformowany do nowego obiektu klasy `DOMDocument`.

[11] *http://www.jclark.com/dsssl/*

Rysunek 8.4. *Klasa XSLTProcessor*



Listing 8.20. *Arkusz stylów XSLT dla katalogu książek*

```
<?xml version="1.0" encoding="iso-8859-2"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" />
  <xsl:template match="/">
    <html>
      <body>
        <ul>
          <xsl:for-each select="books/book">
            <li>
              <xsl:value-of select="author" /><br />
              <xsl:value-of select="title" /><br />
              ISBN: <xsl:value-of select="isbn" /><br />
            </li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Listing 8.21. *Przekształcanie katalogu książek z formatu XML na format HTML wykonywane przez język XSLT*

```
<?php
$dokument    = DOMDocument::load('books.xml');
$arkuszStylów = DOMDocument::load('books.xsl');

$procesor = new XSLTProcessor;
$procesor->importStylesheet($arkuszStylów);

print $procesor->transformToDoc($dokument)->saveXML();
?>
```

```
<?xml version="1.0" standalone"yes"?>
<html>
  <body>
    <ul>
      <li>
        Sebastian Bergmann<br/>
        Profesjonalne tworzenie oprogramowania w PHP 5<br/>
        83-246-0069-8<br/>
      </li>
```

```
      <li>
        Emilian Balanescu<br/>
        PHP5 i MySQL. Zastosowania e-commerce<br/>
        83-7361-830-9<br/>
      </li>
    </ul>
  </body>
</html>
```

Alternatywnym sposobem wykonania tej operacji jest wykorzystanie metod `transformToUri()` i `transformToXML()`, które wynik transformacji dokumentu zapisują w pliku albo zwracają w postaci ciągu znaków.

Parametry arkusza stylów XSL można kontrolować za pomocą metod `getParameter()`, `setParameter()` i `removeParameter()`.

8.5.2. Wywoływanie funkcji PHP z poziomu języka XSLT

Za pomocą funkcji XSL `php:function` można wywoływać dowolne funkcje PHP z wnętrza arkusza stylów XSL. W celu wywołania takiej funkcji koniecznie należy zadeklarować odpowiednią przestrzeń nazw, korzystając przy tym z konstrukcji `xmlns:php="http://php.net/xsl"` (tak jak na listingu 8.22). Z kolei za pomocą metody `registerPHPFunctions()` obiektu procesora XSLT aktywowana jest obsługa wykonywania funkcji PHP wywoływanych z arkuszy stylów XSL (proszę spojrzeć na listing 8.23).

Listing 8.22. *Zastosowanie funkcji PHP w arkuszu stylów XSL*

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:php="http://php.net/xsl" version='1.0'>
  <xsl:template match="/">
    <xsl:value-of select="php:function('str_rot13', 'CUC 5')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 8.23. *Transformacje wykonywane w arkuszu stylów XSL korzystającym z funkcji PHP*

```
<?php
$arkuszStylów = DOMDocument::load('php_function.xsl');

$procesor = new XSLTProcessor;
$procesor->registerPHPFunctions();
$procesor->importStylesheet($arkuszStylów);

$wynik = $procesor->transformToDoc(new DOMDocument);
print $wynik->saveXML();
?>
```

```
PHP 5
```

8.6. Przenoszenie obiektów do sieci WWW za pomocą biblioteki XML_Transformer

Biblioteka *XML_Transformer*[12] (sposób jej instalacji przedstawiony został na rysunku 8.5) wzorowana jest na rozwiązaniach dwóch zadań, które na pierwszy rzut oka nie mają ze sobą nic wspólnego. Po pierwsze, wielu programistów szuka możliwości przeniesienia obiektów PHP „do sieci WWW". Po drugie, przekształcenia danych XML z jednego formatu na inny są nieodłącznym elementem tworzenia aplikacji WWW.

Rysunek 8.5. *Instalowanie biblioteki XML_Transformer*

```
pear install --alldeps XML_Transformer
```

W przeciwieństwie do transformacji XSL rozwiązujących przedstawione wyżej cząstkowe problemy, biblioteka *XML_Transformer* pozwala na proceduralną definicję transformacji XSL. W bibliotece tej można wykorzystać najróżniejsze możliwości PHP, takie jak dostęp do baz danych albo tworzenie grafiki. Dodatkowo, inaczej niż w języku XSLT, wynik działania jednej z operacji pośrednich może być wykorzystany jako wejście kolejnego kroku procesu.

8.6.1. Użytkowanie biblioteki XML_Transformer

Procedury obsługi przestrzeni nazw

Mówiąc ogólnie, kodowanie za pomocą biblioteki *XML_Transformer* polega na wykorzystaniu istniejących procedur obsługi przestrzeni nazw (ang. *namespace handler*) lub na tworzeniu własnych. Na listingu 8.24 przedstawione zostało zastosowanie zawartej w pakiecie procedury obsługi przestrzeni nazw `Image`. Procedura ta stanowi „inteligentną" wersję elementu `<img />`, która automatycznie dopasowuje wartości atrybutów `width` i `height` na podstawie informacji o obrazie uzyskanych za pomocą funkcji `getimagesize()`.

Listing 8.24. *Inteligentny znacznik <img /> z procedurą obsługi przestrzeni nazw Image*

```
<?php
require_once 'XML/Transformer.php';
require_once 'XML/Transformer/Namespace/Image.php';

$t = new XML_Transformer(
  array(
    'debug'                => TRUE,
    'overloadedNamespaces' => array(
      'img' => new XML_Transformer_Namespace_Image
    )
  )
);
```

[12] *http://www.sebastian-bergmann.de/XML_Transformer/*

```
print $t->transform('<img:img src="image.png" />');
?>
```

```
<img height="42" src="image.png" width="23" />
```

W celu wykonania tych operacji najpierw ładowane są klasy `XML_Transformer` i `XML_Transformer_Namespace_Image`. Następnie tworzony jest obiekt klasy `XML_Transformer`, a w jego konstruktorze przekazywana jest przestrzeń nazw `img`, związana z obiektem klasy `XML_Transformer_Namespace_Image`. Faktyczna transformacja wykonywana jest w metodzie `transform`, której w jedynym parametrze przekazywany jest wejściowy element XML w postaci ciągu znaków.

Do ustalania parametrów i definiowania transformacji w klasie `XML_Transformer` udostępniane są odpowiednie metody. Na listingu 8.25 najpierw tworzony jest zatem obiekt klasy `XML_Transformer`, dla którego aktywowany jest tryb debugowania i ładowane są transformacje procedury obsługi przestrzeni nazw.

Listing 8.25. *Ustalanie parametrów za pomocą metod klasy XML_Transformer*

```
<?php
require_once 'XML/Transformer.php';
require_once 'XML/Transformer/Namespace/Image.php';

$t = new XML_Transformer;

$t->setDebug(TRUE);

$t->overloadNamespace(
  'img',
  new XML_Transformer_Namespace_Image
);
?>
```

Parametry transformacji można ustalać również w momencie tworzenia obiektu, przekazując je konstruktorowi klasy `XML_Transformer`, tak jak zrobione to zostało na listingu 8.24.

8.6.2. Stosowanie klasy sterownika XML_Transformer

Klasy sterowników

Biblioteka *XML_Transformer*, oprócz możliwości bezpośredniego korzystania z klasy `XML_Transformer` i wykonywania transformacji za pomocą jej metody `transform`, udostępnia klasy sterowników, rozbudowujących standardowe zachowania tej klasy. Na listingu 8.26 przedstawiony został sposób użycia klasy sterownika `XML_Transformer_Driver_OutputBuffer`, która korzysta z dostępnego w PHP mechanizmu bufora wyjściowego (ang. *output-buffer*), aby zachować w buforze całość danych wyjściowych skryptu, w którym jest używana, a następnie zastosować je jako wejściowy dokument XML.

Listing 8.26. *Zastosowanie sterownika Output-Buffer*

```
<?php
require_once 'XML/Transformer/Driver/OutputBuffer.php';
require_once 'XML/Transformer/Namespace/Image.php';
```

```
$t = new XML_Transformer_Driver_OutputBuffer(
  array(
    'overloadedNamespaces' => array(
      'img' => new XML_Transformer_Namespace_Image
    )
   )
);
?>
<img:img src="image.png" />
```

Inną klasą sterownika jest klasa `XML_Transformer_Driver_Cache`, która zapisuje w buforze wynik wykonanej transformacji. Dzięki temu można pominąć przyszłe transformacje wykonywane na identycznych danych wejściowych i podać gotowy wynik pobrany z takiego bufora.

8.6.3. Dostarczane procedury obsługi przestrzeni nazw

Poznaliśmy już jedną procedurę obsługi przestrzeni nazw zawartą w klasie `XML_Transformer_Namespace_Image`, należącej do biblioteki *XML_Transformer*. Teraz przedstawione zostaną skrócone opisy pozostałych procedur obsługi przestrzeni nazw dostarczanych wraz z biblioteką.

Anchor

Procedura obsługi przestrzeni nazw `Anchor` udostępnia cały szereg znaczników, które normalnie pozwalają na tworzenie pośrednich, nazwanych łączy powiązanych z przestrzenią nazw `a`, czyli tak zwanych URN. Przykład zastosowania tej procedury został przedstawiony na listingu 8.27.

Listing 8.27. *Przykład zastosowania procedury obsługi przestrzeni nazw Anchor*

```
<?php
require_once 'XML/Transformer/Driver/OutputBuffer.php';
require_once 'XML/Transformer/Namespace/Anchor.php';

$t = new XML_Transformer_Driver_OutputBuffer(
  array(
    'overloadedNamespaces' => array(
      'anchor' => new XML_Transformer_Namespace_Anchor
     )
  )
);
?>
<!-- Tworzy nowy wpis -->
<anchor:link name="php" href="http://www.php.net/"
  title="PHP Homepage" />

<!-- Tworzy znacznik HTML <a /> -->
<anchor:iref iref="php">The PHP Homepage</anchor:iref>
```

```
<a href="http://www.php.net/" title="PHP Homepage">The PHP Homepage</a>
```

DocBook

Procedura obsługi przestrzeni nazw `DocBook` udostępnia transformacje pozwalające na przekształcenie podzbioru znaczników *DocBook XML* na format HTML. Można ją traktować jako przykładową kompletną implementację procedury obsługi przestrzeni nazw, w której wykonanie transformacji wymaga wykonania kilku przebiegów. Podobnie jak w procesorze LaTeX, w pierwszym przebiegu zbierane są dane na temat dokumentu, a w kolejnym są one odpowiednio przetwarzane.

Image

Procedura obsługi przestrzeni nazw `Image`, obok prezentowanego już „inteligentnego" znacznika `<img />` udostępnia funkcję dynamicznego generowania grafiki. Działanie i sposób użycia tej funkcji wzorowane są na znaczniku `<gtext />` pochodzącym z platformy Roxen. Przykład użycia tego znacznika przedstawiony został na listingu 8.28.

Listing 8.28. *Przykład zastosowania znacznika <img:gtext />*

```
<?php
require_once 'XML/Transformer/Driver/OutputBuffer.php';
require_once 'XML/Transformer/Namespace/Image.php';

$t = new XML_Transformer_Driver_OutputBuffer(
  array(
    'overloadedNamespaces' => array(
      'img' => new XML_Transformer_Namespace_Image
    )
  )
);
?>
<html>
  <body>
    <img:gtext bgcolor="#ffffff" fgcolor="#000000">
      Tekst graficzny
    </img:gtext>
  </body>
</html>
```

```
<html>
  <body>
    <img alt="Tekst graficzny"
         src="/cache/gtext/51445ca8cca049079648630776939e5.png"
         width="113"
         height="16" />
  </body>
</html>
```

W procedurze obsługi przestrzeni nazw `Image` wykorzystywany jest specjalny bufor, w którym składowane są grafiki wygenerowane przez znaczniki `<img:gtext />`, dzięki czemu nie trzeba ponownie tworzyć grafiki przy kolejnym żądaniu o takich samych parametrach.

PHP

Procedura obsługi przestrzeni nazw PHP pozwala między innymi na włączenie kodu PHP bezpośrednio do dokumentu XML, a także na dynamiczne deklarowanie nowych transformacji. Przykład użycia tej procedury przedstawiony został na listingu 8.29.

Listing 8.29. *Przykład zastosowania procedury obsługi przestrzeni nazw PHP*

```
<?php
require_once 'XML/Transformer_OutputBuffer.php';
require_once 'XML/Transformer/Namespace/PHP.php';

$t = new XML_Transformer_OutputBuffer(
  array(
    'overloadedNamespaces' => array(
      'php' => new XML_Transformer_Namespace_PHP
    )
  )
);
?>
<dl>
  <dd><php:expr>time()</php:expr></dd>
  <php:setvariable name="foo">bar</php:setvariable>
  <dd>foo = <php:getvariable name="foo"/></dd>
</dl>
```

```
<dl>
  <dd>1045740810</dd>
  <dd>foo = bar</dd>
</dl>
```

Widget

Procedura obsługi przestrzeni nazw `Widget` udostępnia między innymi transformacje podobne do wykonywanych przez znaczniki `<obox />` z platformy *Roxen*.

8.6.4. Tworzenie własnej procedury obsługi przestrzeni nazw

W tym punkcie zajmiemy się sposobami przygotowania własnej procedury obsługi przestrzeni nazw. Jak można się było przekonać w podawanych wcześniej przykładach, biblioteka *XML_Transformer* wiąże funkcje PHP z elementami XML, przy czym dana klasa wiązana jest z określoną przestrzenią nazw. Pseudoprzestrzeń nazw `&MAIN` pozwala na podobne operacje bez dowiązywania konkretnej przestrzeni nazw. Klasa PHP wywiedziona z klasy `XML_Transformer_Namespace` dla każdego elementu powiązanej z nią przestrzeni nazw implementuje dwie metody opisujące transformacje. Metoda `start_ELEMENT($attributes)` wywoływana jest z atrybutami zapisanymi w parametrze `$attributes` dla każdego otwierającego znacznika elementu, natomiast metoda `end_ELEMENT($cdata)` wywoływana jest dla znaczników zamykających element, a w jej parametrze przekazywane są dane `CDATA` tego elementu.

Obie metody mogą (ale nie muszą) zwracać fragment danych XML w postaci ciągu znaków, który włączany jest do danych wyjściowych w zastępstwie aktualnie obsługiwanego elementu XML. Trzeba przy tym uważać, aby zwracany fragment nie naruszał prawidłowej budowy dokumentu XML.

Na listingu 8.30 przedstawiona została klasa `WitajSwiecie`, która implementuje taką właśnie procedurę obsługi przestrzeni nazw i za pomocą metod `start_witajSwiecie()` i `end_witajSwiecie()` udostępnia transformację znaczników `<witajSwiecie />`.

Listing 8.30. *Procedury obsługi przestrzeni nazw „Witaj świecie”*

```
<?php
require_once 'XML/Transformer/Driver/OutputBuffer.php';
require_once 'XML/Transformer/Namespace.php';

class WitajSwiecie extends XML_Transformer_Namespace {
  function start_witajSwiecie($atrybuty) {
    return '<html><body>';
  }

  function end_witajSwiecie($cdata) {
    return $cdata . 'Witaj świecie!</body></html>';
  }
}

$t = new XML_Transformer_Driver_OutputBuffer(
  array(
    'overloadedNamespaces' => array(
      '&MAIN' => new WitajSwiecie
    )
  )
);
?>
<witajŚwiecie />
```

```
<html>
  <body>
    Witaj świecie!
  </body>
</html>
```

Klasa `XMLUtil` udostępnia metody, które mogą być bardzo użyteczne w czasie tworzenia aplikacji PHP pracujących z danymi w formacie XML. W czasie programowania procedur obsługi przestrzeni nazw z wykorzystaniem biblioteki *XML_Transformer* najbardziej powinny nas zainteresować następujące metody:

- `attributesToString($attributes)` — zamienia w ciąg znaków tablicę atrybutów XML, na przykład taką, jaka przekazywana jest w parametrze do metody `start_ELEMENT($attributes)`. Przykład użycia tej metody przedstawiony został na listingu 8.31.

Listing 8.31. *Przykład zastosowania metody XML_Util::attributesToString*

```
<?php
require_once 'XML/Util.php';

print XML_Util::attributesToString(
  array(
```

```
    'width'  => 23,
    'height' => 42
  )
);
?>
```

```
height="42" width="23"
```

- `splitQualifiedName($element)` — rozdziela nazwę elementu XML na nazwę przestrzeni nazw i właściwą nazwę elementu oraz zwraca wynik takiej operacji w postaci tablicy. Przykład użycia tej metody przedstawiony został na listingu 8.32.

Listing 8.32. *Przykład zastosowania metody XML_Util::splitQualifiedName*

```
<?php
require_once 'XML/Util.php';

print_r(
  XML_Util::splitQualifiedName('img')
);

print_r(
  XML_Util::splitQualifiedName('img:img')
);
?>
```

```
Array
(
    [0] => &MAIN
    [1] => img
)
Array
(
    [0] => img
    [1] => img
)
```

W czasie tworzenia procedury obsługi przestrzeni nazw należy zawsze pamiętać, że biblioteka *XML_Transformer* wewnętrznie pracuje z parserem SAX. Z tego powodu nie jest możliwe utworzenie w jednym przebiegu spisu zawartości dokumentu XML ani transformacje zmieniające kolejność elementów. Jeżeli wykonanie takich operacji jest mimo wszystko wymagane, to wejściowe dane XML muszą być transformowane w dwóch przebiegach. Wartość `TRUE` wpisana do atrybutu `$secondPassRequired` oznacza, że w czasie transformacji konieczne jest wykonanie dwóch przebiegów.

Tryb debugowania

W czasie tworzenia własnych procedur obsługi przestrzeni nazw biblioteka *XML_Transformer* może wspomagać nas swoim specjalnym trybem debugowania. Tryb ten może być aktywowany poprzez przekazanie odpowiedniego parametru do konstruktora klasy `XML_Transformer` albo bezpośrednie wywołanie metody `setDebug(TRUE)`, tak jak zostało to zrobione na listingach 8.24 i 8.25.

- Dla każdego otwierającego znacznika elementu XML w dzienniku zapisywane są: aktualny poziom stosu elementów, nazwa elementu oraz jego atrybuty.
- Dla każdego zamykającego znacznika elementu XML zapisywane są: aktualny poziom stosu elementów, nazwa elementu oraz zawartość jego obszaru `CDATA`.

Na rysunku 8.6 przedstawione zostały informacje debugowania zapisane dla kodu z listingu 8.24.

```
startElement[1]: img:img  src="image.png"
endElement[1]: img:img (with cdata=)
```

Rysunek 8.6. *Komunikaty debugowania z przykładu dla znacznika <img>*

Przy ustawieniach domyślnych komunikaty debugowania przesyłane są do dziennika błędów serwera WWW. Wypisywanie ich na ekranie jest oczywiście możliwe, z wyjątkiem przypadku zastosowania klasy sterownika `XML_Transformer_Driver_OutputBuffer`.

Błędy występujące w czasie obsługiwania danych XML są zapisywane do dziennika niezależnie od tego, czy tryb debugowania został włączony. Dzięki dostępności zrzutu stosu (ang. stackdump) można określić miejsce w dokumencie XML, w którym nastąpiło naruszenie jego prawidłowej budowy. Takie naruszenie może nastąpić w samym dokumencie XML albo wynikać z jego transformacji. Przykład takiego komunikatu przedstawiony został na rysunku 8.7.

```
Transformer: XML Error: unclosed token at line 1:0
line 1: <img:img src="image.png" />

Stackdump (level: 0) follows:
level=0
element=:
cdata=
```

Rysunek 8.7. *Komunikat o błędzie klasy XML_Transformer zawierający zrzut stosu*