Java i XML. Wydanie III

Autorzy: Brett D. McLaughlin, Justin Edelson
Tłumaczenie: Łukasz Piwko
ISBN: 978-83-246-1011-2
Tytuł oryginału: Java and XML (3rd edition)
Format: B5, stron: 440

Praktyczne zastosowania możliwości języka XML w aplikacji Java

- Z jakich elementów składa się XML?
- W jaki sposób przetwarzać pliki XML w aplikacjach Javy?
- Jak tworzyć własne kanały RSS?

Jesteś programistą Javy i chcesz wykorzystać w swoich aplikacjach technologię XML? Zainteresowała Cię technologia AJAX? Zamierzasz tworzyć własne kanały RSS i podcasty? Java i XML są niemal stworzone do wzajemnej współpracy. W XML tworzy się pliki konfiguracyjne dla aplikacji Javy, zbiory danych i wiele innych elementów. AJAX, bazujący w dużej mierze na języku XML, pozwala na stworzenie eleganckich i wygodnych interfejsów użytkownika dla aplikacji przeglądarkowych. Wiedza o tym, jak efektywnie połączyć XML z Javą, pomoże Ci w budowaniu nowoczesnych programów.

„Java i XML. Wydanie III" to podręcznik, po przeczytaniu którego staniesz się ekspertem w zakresie wykorzystywania możliwości języka XML. Czytając go, poznasz podstawy języka XML, sposoby przetwarzania plików XML w aplikacjach Javy za pomocą API SAX, DOM, StAX, JDOM i dom4j, a także najnowszych wersji JAXP i JAXB. Dowiesz się, jak tworzyć kanały RSS, witryny Web 2.0 i własne podcasty. Przeczytasz także o technologii AJAX i nauczysz się budować z jej wykorzystaniem interfejsy użytkownika dla swoich aplikacji.

- Elementy języka XML
- Walidacja dokumentów XML
- Przetwarzanie plików XML za pomocą SAX
- Wykorzystanie innych technologii do obróbki plików XML
- Wiązanie danych w JAXB
- Tworzenie kanałów RSS
- Transformacja XML na HTML za pomocą JSP
- Korzystanie z mechanizmów AJAX
- Wykorzystanie danych XML w języku ActionScript 3.0

Twórz nowoczesne aplikacje, wykorzystując XML

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 13.

Prezentacyjny XML

Do tej pory XML traktowaliśmy jako technologię niskiego poziomu — użytkownik końcowy nie wie, czy użyte zostały zwykłe pliki właściwości, czy też pliki właściwości XML omawiane w rozdziale 9. Ponadto aplikacje współdzielące omawiane do tej pory dokumenty są z reguły serwerami przetwarzającymi dane. W ostatnim rozdziale przedstawiłem przykłady, w których XML był używany w kontekście klient-serwer: źródła RSS i Atom są dostarczane bezpośrednio do klientów — w tych przypadkach do agregatorów RSS i Atom (oczywiście, istnieją też serwerowe agregatory RSS, takie jak NewsGator — *http://www.newsgator.com* — czy My Yahoo! — *http://my.yahoo.com*). Ale jest to dość ograniczony przypadek, zawężony do specyficznej leksyki RSS i Atom. W niniejszym rozdziale przyjrzymy się bardziej ogólnym przypadkom użycia XML jako części technologii prezentacyjnej w aplikacji sieciowej.

Opracowując ten rozdział przyjąłem kilka założeń. Po pierwsze zakładam, że osoba go czytająca przeczytała też poprzednie rozdziały. Podobnie jak w przypadku biblioteki ROME z poprzedniego rozdziału, będziemy korzystać z niektórych bibliotek, które omówione zostały wcześniej, a w szczególności z DOM. Po drugie zakładam, że Czytelnik dysponuje pewną wiedzą na temat różnych technologii sieciowych, takich jak HTML, JavaScript, serwlety Java i JavaServer Pages (JSP). Ponadto zakładam, że potrafi zainstalować kontener serwletów Javy (np. Apache Tomcat) lub zna kogoś, kto chętnie służy pomocą. W rozdziale tym nie ma informacji na temat pisania aplikacji sieciowych w Javie. Osobom, które nie mają przynajmniej podstawowej wiedzy na temat wymienionych technologii, stanowczo polecam odłożenie tej książki, uzupełnienie wiadomości i wrócenie do niej z odpowiednim przygotowaniem. Do najlepszych książek na temat technologii sieciowych Javy należą *Java Servlet. Programowanie. Wydanie II* autorstwa Jasona Huntera (Helion, Gliwice 2002) i *JavaServer Pages. Leksykon Kieszonkowy* napisana przez Hansa Bergstena (Helion, Gliwice 2002).

XML a wzorzec Model-View-Controller

Kiedy odnoszę się do XML jako technologii prezentacyjnej, to przede wszystkim mam na myśli widok w aplikacji wykorzystujący architekturę Model-View-Controller (MVC). MVC jest architekturą oprogramowania, która początkowo została stworzona jako wzorzec dla tradycyjnych aplikacji klienckich (jak te tworzone za pomocą Swinga), ale została powszechnie adaptowana do aplikacji sieciowych. Krótko mówiąc, zastosowanie MVC pozwala na podział aplikacji na trzy główne obszary:

Model
Surowe dane i reguły biznesowe aplikacji.

View
Dostępna dla użytkownika interpretacja widoku modelu.

Controller
Procedury odbierające żądania od użytkowników, interpretujące je, oddziałujące z modelem i dostarczające widoku z wszystkimi niezbędnymi obiektami modelu.

Jako konkretny przykład może posłużyć aplikacja sieciowa MVC napisana przy użyciu serwletów Javy i JSP, która może przetworzyć żądanie użytkownika w czterech etapach:

1. serwlet (kontroler) odbiera żądanie i przetwarza je,
2. serwlet wywołuje jakieś metody na obiekcie dostępu do danych,
3. serwlet przekazuje obiekty danych modelu do strony JSP w celu ich wizualizacji,
4. strona JSP wysyła na wyjście stronę HTML zawierającą dane z obiektów modelu.

Dostępnych jest wiele środowisk MVC dla Javy, które dostarczają znaczną część podstawowego kodu potrzebnego w każdej aplikacji sieciowej. Do najpopularniejszych należą Apache Struts (*http://struts.apache.org*), Spring MVC (*http://www.springframework.org*), JavaServer Faces (*http://java.sun.com/javaee/javaserverfaces*) oraz Tapestry (*http://tapestry.apache.org*).

XML w aplikacjach sieciowych MVC

XML w aplikacjach sieciowych MVC można użyć w kilku miejscach. Większość tego typu środowisk wykorzystuje XML jako wewnętrzny mechanizm konfiguracyjny. Dla nas bardziej interesujące jest, kiedy XML używa się do przekazywania danych pomiędzy widokiem a kontrolerem. Zamiast do widoku przekazywać jeden lub więcej obiektów modelu, kontroler tworzy reprezentację XML obiektów modelu i tak utworzony dokument przekazuje do widoku. W niektórych przypadkach aplikacja jest odpowiedzialna za dostarczenie XML — widok tylko dokonuje serializacji dokumentu jako odpowiedź HTTP. W innych widok stanowi pewnego rodzaju transformację po stronie serwera XML dostarczonego przez kontroler na XML o innej składni lub HTML. Ponadto transfer danych modelu pomiędzy kontrolerem a widokiem przy użyciu XML pozwala na przeniesienie wszystkich niezbędnych operacji transformujących z serwera do aplikacji użytkownika (z reguły przeglądarki internetowej).

Na początku utworzymy prosty serwlet tworzący dokument XML. Listing 13.1 przedstawia serwlet tworzący dokument XML zawierający listę książek. Nasz model jest listą (`List`) obiektów `Map`. Po utworzeniu obiektu DOM `Document` serwlet wysyła na wyjście dokument przy użyciu techniki transformacji identycznościowej, którą omówiłem w rozdziale 7.

Listing 13.1. Serwlet generujący dokument XML

```
package javaxml3.ch13;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Result;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;

public class BookListXMLServlet extends HttpServlet {

    private DocumentBuilderFactory documentBuilderFactory;

    private TransformerFactory transformerFactory;

    public void init() {
        documentBuilderFactory = DocumentBuilderFactory.newInstance();
        transformerFactory = TransformerFactory.newInstance();
    }

    private Element newElementFromMap(Map map, String key, Document doc) {
        String text = (String) map.get(key);
        Text textNode = doc.createTextNode(text);
        Element element = doc.createElement(key);
        element.appendChild(textNode);
        return element;
    }

    protected void renderDocument(Document doc, HttpServletRequest request,
            HttpServletResponse response) throws IOException, ServletException {

        // Informacja dla przeglądarki, że wysyłany jest XML.
        response.setContentType("text/xml");

        // Transformacja identycznościowa.
        try {
            Transformer identity = transformerFactory.newTransformer();

            // Nasz obiekt Result to StreamResult opakowujący
            // ServletOutputStream.
            Result result = new StreamResult(response.getOutputStream());
            identity.transform(new DOMSource(doc), result);
        } catch (TransformerException e) {
            throw new ServletException(
                    "Nie można wykonać transformacji identycznościowej", e);
        }
    }

    protected void service(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        List bookList = BookListFactory.INSTANCE;

        DocumentBuilder docBuilder = null;
        try {
            docBuilder = documentBuilderFactory.newDocumentBuilder();
```

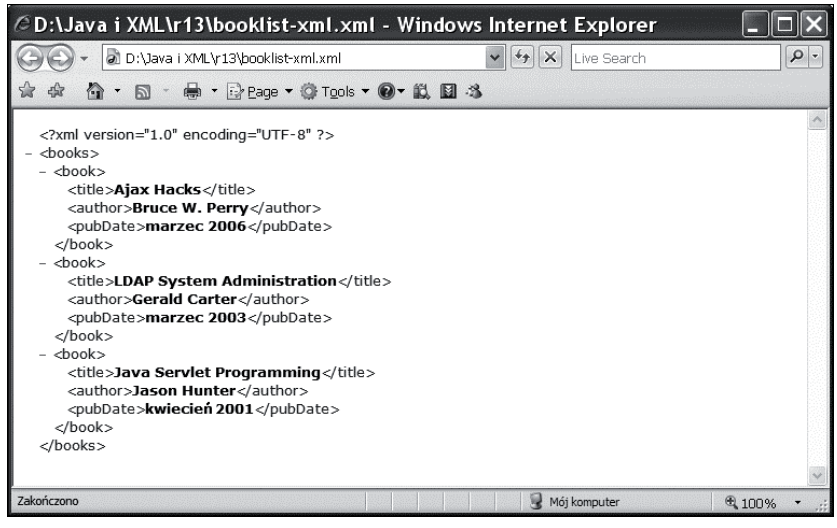
```
        } catch (ParserConfigurationException e) {
            throw new ServletException(
                    "Nie można utworzyć DocumentBuilderFactory", e);
        }

        // Tworzenie dokumentu DOM.
        Document doc = docBuilder.newDocument();
        Element books = doc.createElement("books");
        doc.appendChild(books);
        for (Iterator it = bookList.iterator(); it.hasNext();) {
            Map bookMap = (Map) it.next();
            Element book = doc.createElement("book");
            books.appendChild(book);
            book.appendChild(newElementFromMap(bookMap,
                    BookListConstants.TITLE, doc));
            book.appendChild(newElementFromMap(bookMap,
                    BookListConstants.AUTHOR, doc));
            book.appendChild(newElementFromMap(bookMap,
                    BookListConstants.PUBDATE, doc));
        }

        renderDocument(doc, request, response);
    }
}

class BookListConstants {

    public static final String AUTHOR = "author";

    public static final String PUBDATE = "pubdate";

    public static final String TITLE = "title";
}
class BookListFactory {

    public static final List INSTANCE;

    static {
        List templist = new ArrayList();
        Map m = new HashMap();
        m.put(BookListConstants.TITLE, "Ajax Hacks");
        m.put(BookListConstants.AUTHOR, "Bruce W. Perry");
        m.put(BookListConstants.PUBDATE, "merzec 2006");
        templist.add(m);

        m = new HashMap();
        m.put(BookListConstants.TITLE, "LDAP System Administration");
        m.put(BookListConstants.AUTHOR, "Gerald Carter");
        m.put(BookListConstants.PUBDATE, "marzec 2003");
        templist.add(m);

        m = new HashMap();
        m.put(BookListConstants.TITLE, "Java Servlet Programming");
        m.put(BookListConstants.AUTHOR, "Jason Hunter");
        m.put(BookListConstants.PUBDATE, "kwiecień 2001");
        templist.add(m);

        INSTANCE = Collections.unmodifiableList(templist);
    }

    private BookListFactory() {
    }

}
```

Listing 13.2 zawiera plik konfiguracyjny serwletu *web.xml*, który mapuje ten serwlet na ścieżkę */booklist-xml*.

Listing 13.2. Przykładowy plik web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="ch13-servlet" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <display-name>ch13-servlet</display-name>
    <servlet>
        <servlet-name>BookListXMLServlet</servlet-name>
        <servlet-class>javaxml3.ch13.BookListXMLServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>BookListXMLServlet</servlet-name>
        <url-pattern>/booklist-xml</url-pattern>
    </servlet-mapping>
</web-app>
```

Po utworzeniu aplikacji sieciowej zawierającej tę klasę serwletu i plik konfiguracyjny możemy obejrzeć wyniki w przeglądarce, jak na rysunku 13.1.



Rysunek 13.1. Dane wyjściowe XML serwletu

Do utworzenia tego dokumentu nie trzeba było używać serwletu. To samo można uzyskać za pomocą JSP lub innego języka szablonu (np. Velocity). Zamiast serwletu z listingu 13.1 moglibyśmy uzyskać znacznie prostszy serwlet, jak ten na listingu 13.3, a następnie XML utworzyć w JSP z listingu 13.4.

Listing 13.3. Znacznie prostszy serwlet listy książek

```
package javaxml3.ch13;

import java.io.IOException;
import java.util.List;
```

```
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class BookListXMLJSPServlet extends HttpServlet {

    protected void service(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        List bookList = BookListFactory.INSTANCE;

        request.setAttribute("bookList", bookList);
        RequestDispatcher dispatcher = getServletContext()
                .getRequestDispatcher("/booklist1.jsp");
        dispatcher.include(request, response);
    }

}
```

Listing 13.4. Wizualizacja XML za pomocą znaczników JSP

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@ page contentType="text/xml"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<books>
<c:forEach items="${bookList}" var="book">
    <book>
        <title>${book.title}</title>
        <author>${book.author}</author>
        <pubdate>${book.pubdate}</pubdate>
    </book>
</c:forEach>
</books>
```

Należy zwrócić uwagę, że deklaracja XML znajduje się na samej górze strony JSP, nad dyrektywami. Jest tak dlatego, że deklaracja XML powinna być na samym początku dokumentu XML.

Jednak przy generowaniu XML za pomocą JSP, jak na listingu 13.4, można utworzyć źle sformułowany dokument. Jako że serwlet z listingu 13.2 tworzy dokument w zgodzie z DOM, nie ma możliwości utworzenia niepoprawnie sformułowanego dokumentu (chyba że jest błąd w bibliotekach XML). Przy zastosowaniu JSP jako metody tworzenia dokumentów XML nie ma sposobu na ochronę przed przypadkowym stworzeniem dokumentu zawierającego źle dopasowane znaczniki zamykające, jak poniżej:

```
<?xml version="1.0" encoding="UTF-8" ?>
<books>
    <book>
        <title>Ajax Hacks</title>
        <author>Bruce W. Perry</pubdate>
        <pubdate>marzec 2006</author>
    </book>
</books>
```

Nie da się wyeliminować prawdopodobieństwa wystąpienia jakiegokolwiek błędu związanego z poprawną formulacją. Aczkolwiek ten sposób tworzenia dokumentów za pomocą JSP na pewno jest szybszy i potrzebuje mniej pamięci, ponieważ pomija tworzenie pośrednich obiektów DOM.

Transformacja na HTML za pomocą JSP

Jeśli zamiast XML spodziewaną odpowiedzią od aplikacji sieciowej są dokumenty HTML, to do dyspozycji mamy kilka możliwości transformacji reprezentacji XML obiektów modelu na HTML. Mimo że podstawowy JSP nie udostępnia żadnych funkcji przeznaczonych specjalnie do pracy z XML, to biblioteka Java Standard Tag Library zawiera kilka znaczników przeznaczonych specjalnie dla dokumentów XML. Za ich pomocą można przekształcać dokumenty XML na HTML przy użyciu XPath lub XSLT. Dodatkowo JSLT zawiera znaczniki pozwalające na przetworzenie dokumentu XML na obiekt DOM, do którego można następnie stosować pozostałe znaczniki.

Znaczniki XML JSTL

JSTL to zbiór znaczników JSP, których przeznaczeniem jest zaspokojenie różnych podstawowych potrzeb podczas pisania stron JSP. Zawiera bibliotekę znaczników XML udostępniającą szereg znaczników używanych do przetwarzania dokumentów XML za pomocą stron JSP:

`out`
: Ewaluuje wyrażenie XPath i wysyła wynik na wyjście.

`parse`
: Przetwarza i włącza łańcuch do obiektu DOM `Document`.

`set`
: Ewaluuje wyrażenie XPath i zapisuje wynik jako lokalną zmienną JSP.

`if`
: Wykonuje to, co jest w ciele znacznika, jeśli wyrażenie XPath zwróci wartość `true`.

`choose-when-otherwise`
: Ma podobną funkcjonalność co konstrukcja Javy `switch-case-default`, przy użyciu wyrażeń XPath.

`forEach`
: Przechodzi przez listę wierzchołków DOM.

`transform`
: Wykonuje transformację XSLT.

Jako że wszystkie te znaczniki (poza `parse`) przyjmują jako punkt początkowy wierzchołek DOM, możemy zmodyfikować metodę `renderBooks()` z listingu 13.1 w taki sposób, aby zapisywała obiekt DOM `Document` jako atrybut żądania i wysyłała go do pliku o nazwie *booklist2.jsp*:

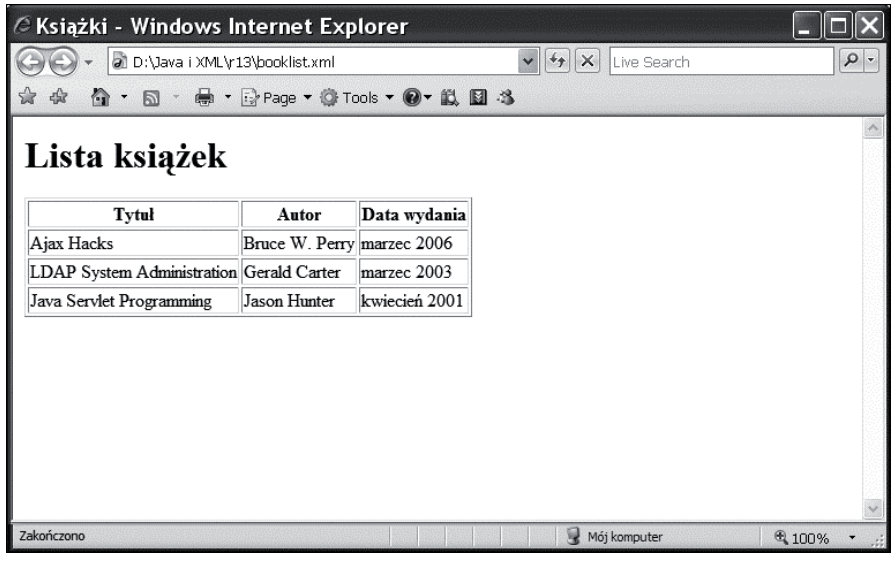
```
private void renderBooks(Document doc, HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {

    request.setAttribute("xml", doc);
    RequestDispatcher dispatcher = getServletContext()
            .getRequestDispatcher("/booklist2.jsp");
    dispatcher.include(request, response);
}
```

Listing 13.5 przedstawia stronę JSP *booklist.jsp*. Należy zwrócić uwagę, że atrybut żądania `xml` jest dostępny jako zmienna XPath. Poza atrybutami żądań w tych wyrażeniach XPath można także wykorzystać atrybuty sesji i aplikacji, parametry żądania, nagłówki HTTP, pliki cookies oraz parametry inicjalizacyjne serwletu.

Listing 13.5. Wysyłanie na wyjście listy książek za pomocą JSTL

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Lista książek</title>
</head>
<body>
<h1>Lista książek</h1>
<table border="1">
    <tbody>
        <tr>
            <th>Tytuł</th>
            <th>Autor</th>
            <th>Data wydania</th>
        </tr>
        <x:forEach select="$xml/books/book" var="book">
            <tr>
                <td><x:out select="$book/title" /></td>
                <td><x:out select="$book/author" /></td>
                <td><x:out select="$book/pubdate" /></td>
            </tr>
        </x:forEach>
    </tbody>
</table>
</body>
</html>
```

Po zaktualizowaniu kodu serwletu i strony JSP wynik HTML jest już znacznie bliższy temu, czego się spodziewaliśmy, co widać na rysunku 13.2.



Rysunek 13.2. Dane książek w formacie HTML wygenerowane przez JSP

Jak już wspominałem powyżej, w JSP jest znacznik transformacji służący do wykonywania przekształceń za pomocą XSL. Przejdźmy zatem do transformacji XSL.

Używanie XSLT

Jak można się było przekonać we wcześniejszych rozdziałach, XSLT jest niezwykle pożytecznym narzędziem do transformacji dokumentów XML z jednego rodzaju składni na inny, jak również na HTML. Jedną z bardziej interesujących opcji dostępnych dzięki XSLT jest przerzucenie ciężaru dokonywania konwersji na aplikacje klienckie, jako że większość współczesnych przeglądarek obsługuje XSLT. Aczkolwiek podejście to ma pewne wady. Jeśli tworzone aplikacje nie są przeznaczone dla bardzo wąskiego grona kontrolowanych odbiorców, będziemy mieli bardzo mały, jeśli w ogóle, wpływ na szybkość działania ich aplikacji klienckich. W wyniku tego różni użytkownicy mogą odnieść całkiem inne wrażenie. Oczywiście, do pewnego stopnia ma to też zastosowanie do zwykłego HTML. Po tym wstępie można tylko dodać, że transformacje po stronie klienta są przydatnym narzędziem w arsenale każdego programisty XML.

Wykonywanie transformacji po stronie klienta

Istnieją dwie metody wykonywania transformacji po stronie klienta — użycie instrukcji przetwarzania i pisanie skryptów wykonywanych po stronie klienta.

Użycie instrukcji przetwarzania

Najprostszym sposobem zażądania transformacji XSL jest umieszczenie instrukcji przetwarzania `xml-stylesheet` pomiędzy deklaracją XML a elementem korzenia. Jeśli przeglądarka otrzyma na przykład dokument z listingu 13.6, to zażąda pliku *http://www.example.com/books.xsl* i wykorzysta zawarty w nim arkusz stylów do przekształcenia dokumentu w celu jego prezentacji.

Listing 13.6. Dokument XML z odniesieniem do arkusza stylów

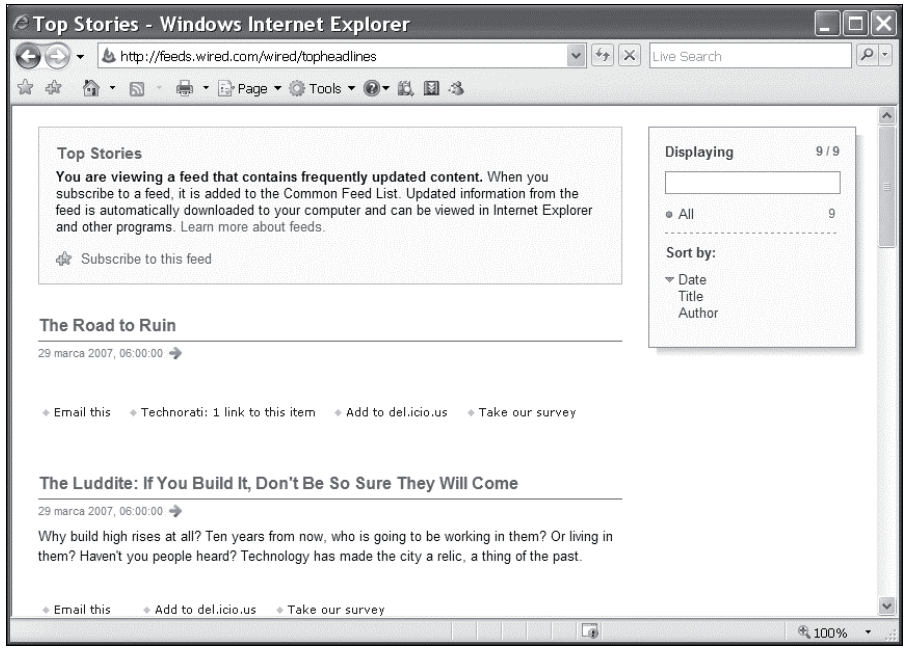
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="http://www.example.com/books.xsl"?>
<books>
  <book>
    <title>Ajax Hacks</title>
    <author>Bruce W. Perry</author>
    <pubDate>marzec 2006</pubDate>
  </book>
  <book>
    <title>LDAP System Administration</title>
    <author>Gerald Carter</author>
    <pubDate>marzec 2003</pubDate>
  </book>
  <book>
    <title>Java Servlet Programming</title>
    <author>Jason Hunter</author>
    <pubDate>kwiecień 2001</pubDate>
  </book>
</books>
```

Instrukcję przetwarzania `xml-stylesheet` można ograniczyć, aby miała zastosowanie tylko w przypadkach, gdy dokument ma być prezentowany w urządzeniach określonego typu. Wiele źródeł RSS wykorzystuje tę instrukcję przetwarzania do wyświetlania stron HTML,

kiedy klientem jest przeglądarka. Przykładowo źródło RSS ze strony *Wired* zawiera poniższą instrukcję przetwarzania:

```
<?xml-stylesheet href="http://feeds.wired.com/~d/styles/rss2full.xsl"
  type="text/xsl" media="screen"?>
```

W wyniku tego, kiedy oglądam to źródło (*http://feeds.wired.com/wired/topheadlines*) w przeglądarce internetowej (która jest widoczna na moim ekranie), widzę jego zawartość oraz różne odnośniki umożliwiające subskrypcję tego źródła w wybranym agregatorze RSS, jak widać na rysunku 13.3.



Rysunek 13.3. Transformacja przeznaczona dla określonego medium

Kiedy ten sam dokument jest ładowany przez agregator RSS, to transformacja jest ignorowana dzięki określeniu w instrukcji przetwarzania odpowiedniego typu mediów.

Transformacje przy użyciu JavaScript

Do uruchamiania procesu transformacji po stronie klienta można używać języka JavaScript zamiast instrukcji przetwarzania w przeglądarkach Mozilla Firefox i Internet Explorer. Więcej na ten temat piszę dalej w podrozdziale „Używanie XSLT z Ajax".

Transformacje po stronie serwera

Jak wcześniej wspominałem, transformacje po stronie klienta mają pewne wady. Poza problemem z wydajnością jest jeszcze to, że nie wszystkie przeglądarki obsługują transformacje. Jednym z możliwych wyjść z takiej sytuacji jest zastosowanie transformacji po stronie klienta, jeśli są przez niego obsługiwane, i po stronie serwera w pozostałych przypadkach.

Transformacja w filtrze

Przy użyciu filtra serwletów Javy można przechwycić żądania dokumentów XML wysyłane przez przeglądarki, które nie obsługują transformacji i, jeśli dokument zawiera instrukcję przetwarzania `xml-stylesheet`, wykonać transformację na serwerze. Filtr taki, który wykonuje transformacje po stronie serwera, obsługując tekstową przeglądarkę Lynx, przedstawiono na listingu 13.7.

Listing 13.7. Transformacja w filtrze

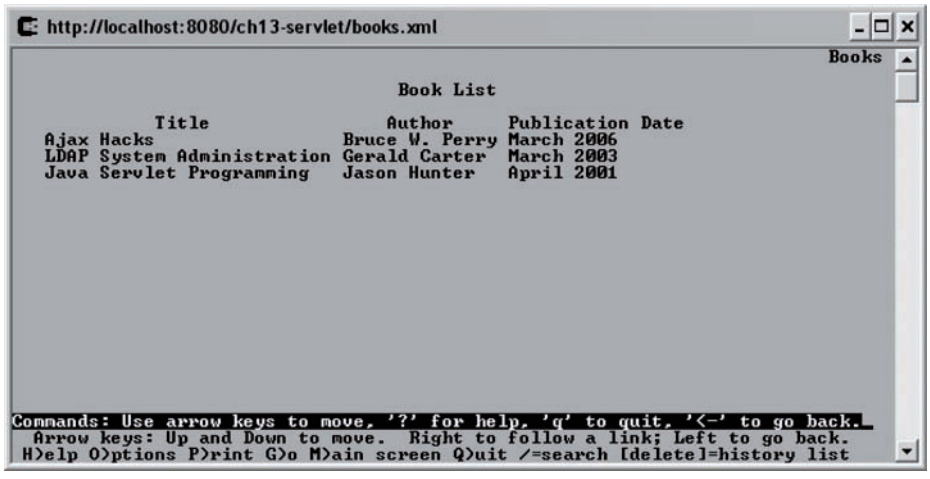
```
package javaxml3.ch13;

import java.io.CharArrayWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StringReader;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class XSLFilter implements Filter {

    private Pattern hrefPattern;

    private SAXParserFactory saxParserFactory;

    private TransformerFactory transformerFactory;

    private Pattern typePattern;

    public void destroy() {
    }

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        if (request instanceof HttpServletRequest) {
            if (isLynxRequest((HttpServletRequest) request)) {
                CharResponseWrapper wrapper = new CharResponseWrapper(
                        (HttpServletResponse) response);
                chain.doFilter(request, wrapper);
                String xml = wrapper.toString();
```

```
            // Przetwarzaj tylko odpowiedzi XML.
            if ("text/xml".equals(wrapper.getContentType())) {
                String stylesheetHref = getStylesheetHref(xml);
                if (stylesheetHref != null) {
                    response.setContentType("text/html");
                    transformResponse(response, xml, stylesheetHref);
                } else {
                    response.getWriter().print(xml);
                }
            } else {
                response.getWriter().print(xml);
            }
        } else {
            chain.doFilter(request, response);
        }
    } else {
        chain.doFilter(request, response);
    }
}

private String getStylesheetHref(String xml) {
    StylesheetPIHandler handler = new StylesheetPIHandler();
    try {
        SAXParser parser = saxParserFactory.newSAXParser();
        InputSource input = new InputSource(new StringReader(xml));
        parser.parse(input, handler);

        return handler.getHref();
    } catch (Exception e) {
        return null;
    }
}

public void init(FilterConfig config) throws ServletException {
    saxParserFactory = SAXParserFactory.newInstance();
    transformerFactory = TransformerFactory.newInstance(  );
    typePattern = Pattern.compile(".*type\\s*=\\s*\"(\\S*)\".*");
    hrefPattern = Pattern.compile(".*href\\s*=\\s*\"(\\S*)\".*");
}

private boolean isLynxRequest(HttpServletRequest request) {
    String userAgent = request.getHeader("User-Agent");
    return (userAgent.indexOf("Lynx") >= 0);
}

private void transformResponse(ServletResponse response, String xml,
        String stylesheetHref) throws IOException, ServletException {
    StreamSource sheetSource = new StreamSource(stylesheetHref);
    StreamSource xmlSource = new StreamSource(new StringReader(xml));
    StreamResult result = new StreamResult(response.getWriter());
    try {
        Transformer trans = transformerFactory.newTransformer(sheetSource);
        trans.transform(xmlSource, result);
    } catch (TransformerException e) {
        throw new ServletException("Nie można wykonać transformacji.", e);
    }
}

class CharResponseWrapper extends HttpServletResponseWrapper {
    private CharArrayWriter output;
```

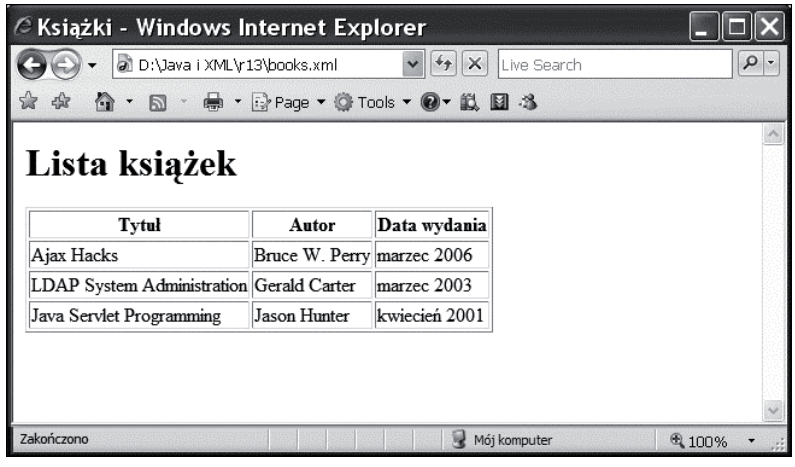
```
        public CharResponseWrapper(HttpServletResponse response) {
            super(response);
            output = new CharArrayWriter();
        }

        public ServletOutputStream getOutputStream() throws IOException {
            return new ServletOutputStream() {

                public void write(int b) throws IOException {
                    output.write(b);
                }

            };
        }

        public PrintWriter getWriter() {
            return new PrintWriter(output);
        }

        public String toString() {
            return output.toString();
        }
    }

    class StylesheetPIHandler extends DefaultHandler {

        private String href = null;

        public String getHref() {
            return href;
        }

        public void processingInstruction(String target, String data)
                throws SAXException {
            if ("xml-stylesheet".equals(target)) {
                Matcher typeMatcher = typePattern.matcher(data);
                if (typeMatcher.matches()
                        && "text/xsl".equals(typeMatcher.group(1))) {
                    Matcher hrefMatcher = hrefPattern.matcher(data);
                    if (hrefMatcher.matches())
                        href = hrefMatcher.group(1);
                }
            }
        }
    }
}
```

Jak widać na listingu, filtr ten przechwytuje żądania. Jeśli zaistnieje możliwość, że trzeba będzie dane wysyłane w odpowiedzi przetransformować, filtr zamieni bieżący obiekt `HttpServlet-Response` na obiekt osłonowy buforujący te dane. Po wygenerowaniu danych odpowiedzi parser SAX szuka instrukcji przetwarzania, a następnie dwa wyrażenia regularne pozwalają na sprawdzenie odpowiedniego identyfikatora URI arkusza stylów.

Po zastosowaniu tego filtra w wyniku żądania pliku *books.xml* do przeglądarki Lynx zostanie zwrócony kod HTML, a do innych przeglądarek XML. Rysunki 13.4 i 13.5 przedstawiają ten sam kod w przeglądarkach Lynx i Internet Explorer. Jedyna różnica polega na tym, gdzie ten kod HTML został wygenerowany.



Rysunek 13.4. Wynik transformacji po stronie serwera w przeglądarce Lynx



Rysunek 13.5. Wynik transformacji po stronie klienta w przeglądarce Internet Explorer

Transformacje przy użyciu JSTL

Jak wcześniej wspominałem, JSTL udostępnia znacznik `transform`, który pozwala przekształcać dokumenty XML. Do tego potrzebne są dokument XML i arkusz stylów. Można je zdefiniować w kodzie strony:

```
<c:set var="doc">
    <books>
        <book>
            <title>Ajax Hacks</title>
            <author>Bruce W. Perry</author>
            <pubdate>marzec 2006</pubdate>
        </book>
        <!-- Dodaj tu więcej książek. -->
```

```
    </books>
</c:set>
<c:set var="xsl">
    <xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="html" />
        <xsl:template match="books">
            <html>
            <head>
            <title>Książki</title>
            </head>
            <body>
            <h1>Lista książek</h1>
            <table border="1">
                <tbody>
                    <tr>
                        <th>Tytuł</th>
                        <th>Autor</th>
                        <th>Data wydania</th>
                    </tr>
                    <xsl:apply-templates select="book" />
                </tbody>
            </table>
            </body>
            </html>
        </xsl:template>
        <xsl:template match="book">
            <tr>
                <td><xsl:value-of select="title" /></td>
                <td><xsl:value-of select="author" /></td>
                <td><xsl:value-of select="pubdate" /></td>
            </tr>
        </xsl:template>
    </xsl:stylesheet>
</c:set>

<x:transform doc="${doc}" xslt="${xsl}" />
```

Dokumenty te mogą zostać załadowane z podanych adresów URL:

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<c:import var="doc" url="http://localhost:8080/ch13-servlet/books.xml"/>
<c:import var="xsl" url="http://localhost:8080/ch13-servlet/books.xsl"/>

<x:transform doc="${doc}" xslt="${xsl}" />
```

Kod XML można zamknąć w znaczniku `transform`:

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<c:import var="xsl" url="http://localhost:8080/ch13-servlet/books.xsl"/>

<x:transform xslt="${xsl}">
    <books>
        <book>
            <title>Ajax Hacks</title>
            <author>Bruce W. Perry</author>
            <pubdate>marzec 2006</pubdate>
        </book>
        <!-- Dodaj tu więcej książek. -->
    </books>
</x:transform>
```

Na koniec wynik transformacji można zapisać jako obiekt DOM, względem którego mogą być ewaluowane wyrażenia XPath:

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<c:import var="xsl" url="http://localhost:8080/ch13-servlet/books.xsl"/>

<x:transform xslt="${xsl}" var="result">
    <books>
        <book>
            <title>Ajax Hacks</title>
            <author>Bruce W. Perry</author>
            <pubdate>marzec 2006</pubdate>
        </book>
        <!-- Dodaj tu więcej książek. -->
    </books>
</x:transform>

<x:out select="$result//h1"/>
```

Ajax

Za nazwą Ajax kryje się grupa powiązanych ze sobą wzorców programowania sieciowego wykorzystywanych do tworzenia interaktywnych aplikacji sieciowych. W tradycyjnych, nieużywających technologii Ajax, aplikacjach sieciowych użytkownik odwiedza poszczególne strony jedna po drugiej — każda jego czynność powoduje wysłanie przez przeglądarkę żądania nowej strony. W Ajax natomiast uaktualniany jest tylko pewien fragment strony na podstawie niewielkich pakietów danych, przesyłanych asynchronicznie pomiędzy serwerem a przeglądarką. Dzięki temu aplikacja jest znacznie bardziej interaktywna i zużywa o wiele mniej transferu. Mimo iż termin „Ajax" został ukuty w 2005 roku[1], techniki tego typu były stosowane od chwili wprowadzenia ramek do HTML w połowie lat 90. Początkowo dokumenty HTML ładowano za pomocą JavaScript do ukrytych ramek, a w Internet Explorerze do znacznika `IFRAME`. Dokumenty te zawierały kod JavaScript, który zmieniał wygląd stron. Ponadto dodatkową funkcjonalność uzyskiwano przy użyciu apletów w Javie i wtyczek do przeglądarek.

Mimo że ramki oferowały możliwości asynchroniczne, to były one jednak bardzo podatne na błędy. Na przykład kliknięcie przez użytkownika przycisku przejścia do poprzedniej strony mogło spowodować wejście przeglądarki w nieprawidłowy stan. Techniki alternatywne do ramek, takie jak aplety Javy czy wtyczki, były obciążone własnymi problemami z kompatybilnością i bezpieczeństwem. W wyniku tego aplikacje Ajax nie były zbyt popularne i stosowano je tylko dla określonych grup użytkowników lub platform. Wszystko zmieniło się, kiedy Microsoft wprowadził obiekt `XMLHttpRequest` w przeglądarce Internet Explorer 5 w 2000 roku.

[1] Patrz *http://www.adaptivepath.com/publications/essays/archives/000385.php*.

Czy Ajax to akronim?

Ściśle mówiąc, Ajax jest definiowany jako akronim zbudowany z pierwszych liter słów Asynchronous JavaScript and XML. Jednak przyjął się zapis z tylko pierwszą wielką literą — „Ajax" zamiast „AJAX". Ponadto, w miarę jak wzorce Ajaksa zyskiwały coraz to więcej użytkowników wśród programistów sieciowych, pojawiły się techniki pozwalające na uzyskanie oferowanej przez niego funkcjonalności bez używania XML. Na przykład w aplikacjach Ajax Google (Gmail, Google Maps, Google Spreadsheets) użyto względnie mało XML. W związku z tym coraz bardziej staje się jasne, że Ajax nie powinien być uważany za akronim, a jeśli już, to jego źródła należy doszukiwać się w słowach Asynchronous JavaScript and the `XMLHttpRequest` object.

Obiekt XMLHttpRequest

Obiekt `XMLHttpRequest` jest używany przez języki skryptowe w przeglądarce (zazwyczaj JavaScript, ale nie tylko) do wysyłania żądań HTTP. Jak jego nazwa wskazuje, został stworzony do tworzenia żądań dokumentów XML. Jednak w zachowaniu tego obiektu nie ma nic, co byłoby ściśle związane z XML. Obiekt `XMLHttpRequest` nie został jeszcze ustandaryzowany, ale obecne jego implementacje oferują zbliżone do siebie możliwości. Konsorcjum W3C podjęło już działania w kierunku standaryzacji interfejsu i funkcjonalności tego obiektu. Postęp prac można obejrzeć na stronie *http://www.w3.org/TR/XMLHttpRequest*.

W przeglądarkach Internet Explorer w wersjach wcześniejszych niż 7. obiekt `XMLHttpRequest` jest kontrolką ActiveX, której egzemplarz tworzy się następująco:

```
var req = new ActiveXObject("Microsoft.XMLHTTP");
```

W Internet Explorer 7 i innych przeglądarkach, jest on obiektem JavaScript, którego egzemplarze tworzone są za pomocą słowa kluczowego `new`:

```
var req = new XMLHttpRequest();
```

W związku z tym, aby utworzyć właściwy obiekt dla danej przeglądarki, z reguły wykorzystuje się instrukcję warunkową `if-else`:

```
var req;
if (window.XMLHttpRequest) {
  req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
  req = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Obiekt `XMLHttpRequest` może pracować zarówno w trybie synchronicznym (skrypt oczekuje na odpowiedź od serwera), jak i asynchronicznym (skrypt jest wykonywany po wysłaniu żądania, a po nadejściu odpowiedzi wywoływana jest metoda zwrotna). Listing 13.8 przedstawia stronę, która żąda dokumentu XML i zwraca go w postaci wartości pola formularza osadzonego na stronie HTML.

Listing 13.8. Synchroniczne wykorzystanie obiektu XMLHttpRequest

```
<html>
<body>
<form id="form"><textarea rows="20" cols="80" id="xmlarea"></textarea>
</form>
```

```
<script language="JavaScript">
var req;
var xmlarea = document.forms['form'].elements['xmlarea']
if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
}

if (req == undefined) {
    xmlarea.value = "Obiekt XMLHttpRequest jest niedostępny.";
} else {
    req.open("GET", "/ch13-servlet/booklist-xml", false);
    // Trzeci param = false oznacza tryb synchroniczny.
    req.send(null);
    // Sprawdzenie statusu.
    if (req.status == 200)
        xmlarea.value = req.responseText;
    else
        xmlarea.value = "Got Response Code: " + req.status;
}
</script>
</body>
</html>
```

Kod z listingu 13.9 robi dokładnie to samo, ale pracuje w trybie asynchronicznym.

Listing 13.9. Asynchroniczne zastosowanie obiektu XMLHttpRequest

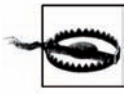
```
<html>
<body>
<form id="form"><textarea rows="20" cols="80" id="xmlarea"></textarea>
</form>
<script language="JavaScript">
var xmlarea = document.forms['form'].elements['xmlarea'];
var req;
if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
}

if (req == undefined) {
    xmlarea.value = "Obiekt XMLHttpRequest jest niedostępny.";
} else {
    req.open("GET", "/ch13-servlet/booklist-xml", true);
    // Trzeci param = true oznacza tryb asynchroniczny.
    req.onreadystatechange = docLoaded;
    req.send(null);
}

function docLoaded() {
    if (req.readyState == 2) {
        xmlarea.value = "Czekaj...";
    }

    if (req.readyState == 4) {
        xmlarea.value = "Otrzymano odpowiedź...";
        // Tylko dla zwiększenia efektu, uaktualnij pole tekstowe po odczekaniu jednej sekundy.
        setTimeout("updateTextArea()",1000);
    }
}

function updateTextArea() {
```

```
    if (req.status == 200)
        xmlarea.value = req.responseText;
    else
        xmlarea.value = "Otrzymano kod odpowiedzi: " + req.status;
}
</script>
</body>
</html>
```

W trybie asynchronicznym obiekt `XMLHttpRequest` musi mieć funkcję zwrotną ustawioną przy użyciu właściwości `onreadystatechange`. Funkcja ta sprawdza wartość właściwości `readyState` w celu określenia bieżącego stanu żądania. Możliwe wartości tej właściwości przedstawia tabela 13.1. Należy zwrócić uwagę, że ta funkcja zwrotna nie może być wywoływana dla każdego z tych stanów.

Tabela 13.1. Dostępne wartości właściwości readyState obiektu XMLHttpRequest

Wartość	Znaczenie
0	Niezainicjalizowane
1	Otwarte
2	Wysłane
3	Otrzymywanie
4	Załadowane



Generalnie obiekt `XMLHttpRequest` nie może wysyłać żądań do serwera innego niż ten, z którego została załadowana zawierająca go strona.

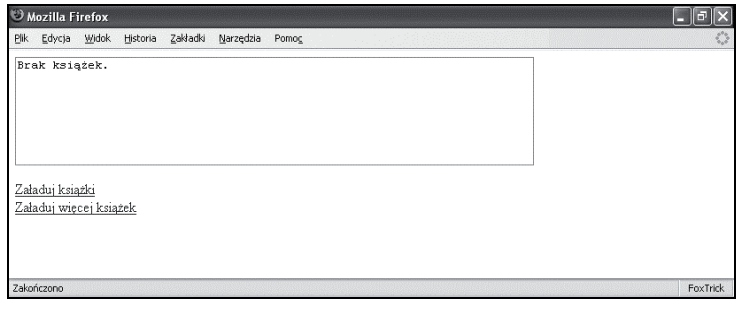
W przykładach tych zawartość dokumentu XML jest dołączana po prostu do strony przy jej pierwszym załadowaniu. Kod na listingu 13.10 idzie o krok dalej i pozwala na przełączanie pomiędzy dwoma dokumentami bez ponownego wczytywania strony.

Listing 13.10. Żądanie różnych dokumentów na podstawie działań użytkownika

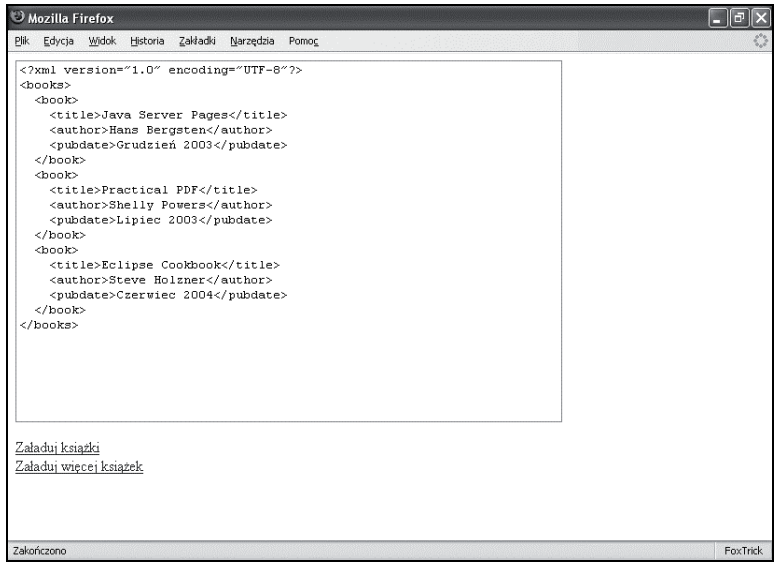
```
<html>
<body>
<form id="form"><textarea rows="20" cols="80" id="xmlarea">Brak książek.</textarea>
</form>
<script language="JavaScript">
var xmlarea = document.forms['form'].elements['xmlarea'];
var req;
if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
}

function loadBooks(filename) {
    if (req == undefined) {
        xmlarea.value = "Obiekt XMLHttpRequest jest niedostępny.";
    } else {
        xmlarea.value = "Czekaj...";
        req.open("GET", "/ch13-servlet/"+filename, true);
        req.onreadystatechange = docLoaded;
        req.send(null);
    }
}
```

```
    if (req.readyState == 4) {
        if (req.status == 200)
            xmlarea.value = req.responseText;
        else
            xmlarea.value = "Got Response Code: " + req.status;
    }
</script>
<a href="javascript:loadBooks('books.xml')">Załaduj książki</a><br/>
<a href="javascript:loadBooks('morebooks.xml')">Załaduj więcej książek</a><br/>
</body>
</html>
```

Przy pierwszym załadowaniu tej strony pole tekstowe zawiera tylko łańcuch `Brak książek`. Wtedy użytkownik może kliknąć jeden z odnośników, aby zapełnić to pole. Na rysunkach odpowiednio 13.6 i 13.7 przedstawiono stronę bezpośrednio po załadowaniu i po kliknięciu odnośnika *Załaduj więcej książek*.



Rysunek 13.6. Strona bezpośrednio po załadowaniu



Rysunek 13.7. Strona po kliknięciu odnośnika Załaduj więcej książek

Używanie DOM z Ajaksem

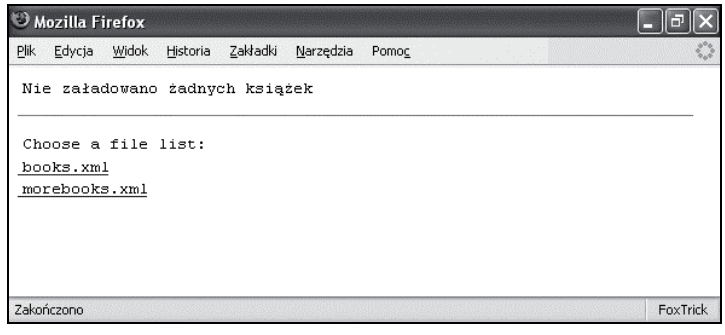
Poza właściwością `responseText` zawierającą odpowiedź w postaci tekstu obiekt `XMLHttpRequest` posiada też właściwość, która zawiera odpowiedź w postaci obiektu DOM. Zamiast tylko umieścić na stronie kod XML (jak było robione wcześniej), można użyć API DOM i wydobyć wartości z tego obiektu. Strona HTML z listingu 13.11 zawiera znacznik `div`, który jest zapełniany tytułami i autorami wszystkich książek z pliku XML. Kiedy zostanie kliknięty jeden z odnośników wyboru pliku, domyślny tekst zostanie nadpisany przez listę książek przy użyciu właściwości `innerHTML`. W kodzie wykorzystana została metoda DOM `getElementByTagName()` oraz różne właściwości tego API.

Listing 13.11. Dostęp do dokumentu DOM w JavaScript

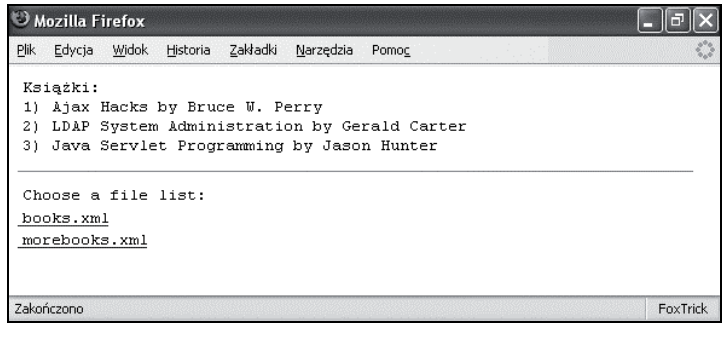
```
<html>
<body>
<div id="xmlarea">
Nie załadowano żadnych książek.
</div>
<hr/>
<script language="JavaScript">
var xmlarea = document.getElementById('xmlarea');
var req;
if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
}

function loadBooks(filename) {
    if (req == undefined) {
        xmlarea.innerHTML = "Obiekt XMLHttpRequest jest niedostępny.";
    } else {
        xmlarea.innerHTML = "Czekaj...";
        req.open("GET", "/ch13-servlet/"+filename, true);
        req.onreadystatechange = docLoaded;
        req.send(null);
    }
}

function docLoaded() {
    if (req.readyState == 4) {
        doc = req.responseXML;
        newValue = "Książki:<br/>\n";
        books = doc.getElementsByTagName("book");
        for (i = 0; i < books.length; i++) {
            book = books.item(i);
            var title;
            var author;
            for (j = 0; j < book.childNodes.length; j++) {
                node = book.childNodes[j];
                if (node.nodeName == "title") {
                    title = node.firstChild.nodeValue;
                } else if (node.nodeName == "author") {
                    author = node.firstChild.nodeValue;
                }
            }

            newValue = newValue + (i+1) + ") " + title +
                " by " + author + "<br/>\n";
        }
```

```
            xmlarea.innerHTML = newValue;
        }
    }
    </script>
    Choose a file list:<br/>
    <a href="javascript:loadBooks('books.xml')">books.xml</a><br/>
    <a href="javascript:loadBooks('morebooks.xml')">morebooks.xml</a><br/>

    </body>
    </html>
```

Rysunek 13.8 przedstawia stronę po jej pierwszym załadowaniu. Strona po załadowaniu pliku *books.xml* została przedstawiona na rysunku 13.9. Warto zwrócić uwagę, że rozmiar elementu `div` został automatycznie zmieniony i w wyniku tego odnośniki przesunęły się w dół.



Rysunek 13.8. Strona DOM przed wybraniem pliku



Rysunek 13.9. Strona DOM po wybraniu pliku

Osoby, które dobrze poznały API DOM w Javie, powinny być w stanie zastosować tę wiedzę do używania DOM w JavaScript. Dowiązania DOM dla języka ECMAScript (standardowego języka skryptowego, od którego pochodzi JavaScript) zostały zdefiniowane przez W3C na stronie *http://www.w3.org/TR/DOM-Level-3-Core/ecma-script-binding.html*. Więcej szczegółów na temat API DOM JavaScript można znaleźć w książce Davida Flanagana pod tytułem *JavaScript: The Definitive Guide* (O'Reilly).

Używanie XSLT z Ajaksem

Zarówno Internet Explorer, jak i Mozilla Firefox obsługują transformacje XSL wykonywane poprzez JavaScript. Niestety, podobnie jak w przypadku obiektu `XMLHttpRequest`, do obsługi transformacji dostęp uzyskuje się w nich na różne sposoby. W Internet Explorerze obiekty DOM mają metodę `transformNode()`, której przekazuje się obiekt DOM `Document` zawierający arkusz stylów:

```
transformOutput = element.transformNode(xslDocument);
```

Wynikiem transformacji może być łańcuch, jak powyżej, lub obiekt DOM.

W Firefoksie jest obiekt JavaScript o nazwie `XSLTProcessor`, do którego dodawane są arkusze stylów przed transformacją:

```
xsltProcessor = new XSLTProcessor();
xsltProcessor.importStylesheet(xslDocument);
transformOutputDoc = xsltProcessor.transformToFragment(element, document);
```

Wynik transformacji w Firefoksie może być fragmentem drzewa DOM albo dokumentem DOM.

Strona na listingu 13.12 w następnym podrozdziale posiada taką samą funkcjonalność przy użyciu XSLT, jaką miała ta z listingu 13.11 z DOM. W tym przykładzie zobaczymy różne sposoby użycia wyniku transformacji w Internet Explorerze i Firefoksie.

```
<html>
<body>
<div id="xmlarea">
Nie załadowano żadnych książek.
</div>
<hr/>
<script language="JavaScript">
var xmlarea = document.getElementById('xmlarea');
var req;
if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
}

req.open("GET", "/ch13-ajax/authortitle.xsl", false);
req.send(null);
var xsldoc = req.responseXML;

function loadBooks(filename) {
    if (req == undefined) {
        xmlarea.innerHTML = "Obiekt XMLHttpRequest jest niedostępny.";
    } else {
        xmlarea.innerHTML = "Czekaj...";
        req.open("GET", "/ch13-servlet/"+filename, true);
        req.onreadystatechange = docLoaded;
        req.send(null);
    }
}

function docLoaded() {
    if (req.readyState == 4 && req.status == 200) {
        xmldoc = req.responseXML;
        if (window.ActiveXObject) {
            xmlarea.innerHTML = xmldoc.transformNode(xsldoc);
        } else if (window.XSLTProcessor) {
            processor = new XSLTProcessor();
```

```
            processor.importStylesheet(xsldoc);
            xmloutput = processor.transformToFragment(xmldoc, document);
            xmlarea.innerHTML = "";
            xmlarea.appendChild(xmloutput);
        } else {
            xmlarea.innerHTML = "XSLT nie jest dostępny w Twojej przeglądarce.";
        }
    }
}
</script>
Choose a file list:<br/>
<a href="javascript:loadBooks('books.xml')">books.xml</a><br/>
<a href="javascript:loadBooks('morebooks.xml')">morebooks.xml</a><br/>

</body>
</html>
```

Wysyłanie XML do serwera

W powyższych przykładach wszystkie żądania wysyłane za pomocą obiektu `XMLHttpRequest` były zwykłymi adresami URL. Można do nich z łatwością dodać opcje w postaci parametrów adresu strony, na przykład *http://www.example.com/servlets/stock?s=IBM*. W przypadku bardziej złożonych parametrów `XMLHttpRequest` pozwala wysyłać treść w ciele żądania przy użyciu metody HTTP `POST`. Jednym z zastosowań tej możliwości jest wysyłanie dokumentów XML do aplikacji działającej na serwerze. W tym celu należy ustawić nagłówek żądania `Content-Type` na `text/xml` i dokument XML przekazać jako łańcuch znaków albo obiekt DOM do metody `send()` obiektu `XMLHttpRequest`.

Tworzenie dokumentu XML w JavaScript

Aby wygenerować dokument XML dla żądania, można zastosować konkatenację łańcuchów, jak poniżej:

```
req.send("<search><name>" + document.getElementById("name").value +
    "</name><year>" + document.getElementById("year").value +
    "</year></search>");
```

Kod JavaScript budujący dokumenty za pomocą łączenia (konkatenacji) łańcuchów może stać się skomplikowany i podatny na błędy, ponieważ nie ma w nim żadnego mechanizmu zapobiegającego tworzeniu źle sformułowanych dokumentów. Aby mieć pewność, że tworzone dokumenty będą poprawne składniowo, należy skorzystać z możliwości DOM dostępnych w JavaScript.

Podobnie jak w przypadku obiektu `XMLHttpRequest`, obiekty DOM `Document` w Internet Explorerze są tworzone jako kontrolki ActiveX, a w innych przeglądarkach poprzez natywne API JavaScript. Aby w przeglądarce Internet Explorer utworzyć obiekt `Document`, trzeba wpierw utworzyć pusty dokument, a następnie w nim element korzenia:

```
doc = new ActiveXObject("Microsoft.XMLDOM");
doc.appendChild(doc.createElement("search"));
```

W Firefoksie i pozostałych przeglądarkach dokument jest tworzony od razu z elementem korzenia:

```
doc = document.implementation.createDocument(null, "search", null);
```

Metoda `createDocument()` przyjmuje identyfikator URI przestrzeni nazw i DTD oraz nazwę elementu korzenia.

Wysyłanie dokumentu XML

Po utworzeniu dokumentu XML jako łańcucha lub obiektu DOM przekazuje się go do metody `send()` obiektu `XMLHttpRequest`. Listing 13.12 przedstawia prosty formularz HTML, w którym naciśnięcie przycisku `submit` spowoduje wysłanie zawartości pól jako dokumentu XML.

Listing 13.12. Wysyłanie danych z formularza jako XML

```
<html>
<body>
<h1>Składnica książek</h1>
<form id="form" onsubmit="search(); return false;">
    Nazwa: <input type="text" id="name"/><br/>
    Rok (optional): <select id="year">
        <option value="notselected">...wybierz...</option>
        <option value="2001">2001</option>
        <option value="2002">2002</option>
        <option value="2003">2003</option>
        <option value="2004">2004</option>
        <option value="2005">2005</option>
    </select><br/>
    <input type="submit" value="Wyślij" />
    <hr/>
    Wyniki wyszukiwania:<br/>
    <textarea rows="3" cols="50" id="results" readonly="true"></textarea>
</form>
<script language="JavaScript">
function search() {
    var req;
    var form = document.forms['form'];
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }

    if (req == undefined) {
        document.getElementById("results").value =
            "Obiekt XMLHttpRequest jest niedostępny.";
    } else {
        req.open("POST", "/ch13-ajax/post", false);
        req.setRequestHeader("Content-Type", "text/xml");

        postDoc = makePostData();

        req.send(postDoc);
        // Sprawdź status.
        if (req.status == 200)
            document.getElementById("results").value = req.responseText;
        else
            document.getElementById("results").value = "Otrzymano kod odpowiedzi: " +
                req.status;
    }
}

function makePostData() {
    var doc;
    if (window.ActiveXObject) {
        doc = new ActiveXObject("Microsoft.XMLDOM");
        doc.appendChild(doc.createElement("search"));
    } else {
        doc = document.implementation.createDocument(null, "search", null);
```

```
    }
    nameElement = doc.createElement("name");
    nameText = doc.createTextNode(document.getElementById("name").value);
    nameElement.appendChild(nameText);
    doc.documentElement.appendChild(nameElement);

    yearElement = doc.createElement("year");
    yearText = doc.createTextNode(document.getElementById("year").value);
    yearElement.appendChild(yearText);
    doc.documentElement.appendChild(yearElement);

    return doc;
}
</script>
</body>
</html>
```

W prawdziwej aplikacji trzeba by było dodać jeszcze mechanizm walidacji w JavaScript wprowadzonych danych przed wysłaniem żądania. Jedną z zalet Ajaksa w porównaniu z tradycyjnymi aplikacjami sieciowymi jest to, że za jego pomocą można przynajmniej część walidacji wykonać przed wysłaniem danych z formularza.

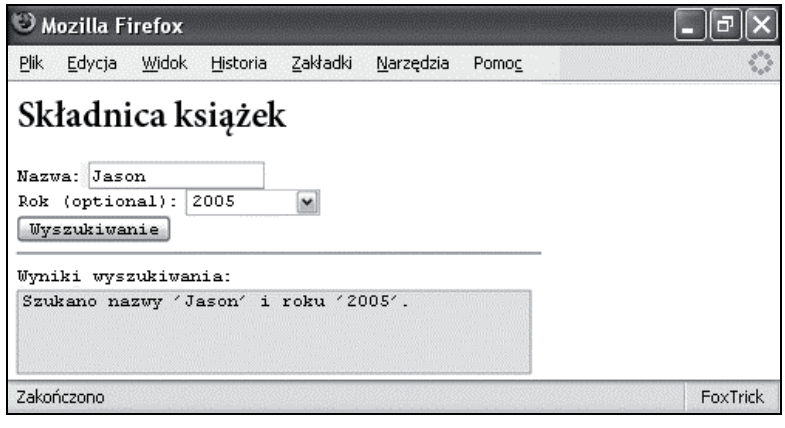
Przy tworzeniu serwletu odbierającego te żądania i analizującego składnię przesyłanego dokumentu można użyć albo obiektu `Reader`, albo `InputStream` z żądania, do których dostęp dają odpowiednio metody `getReader()` i `getInputStream()`. Listing 13.13 zawiera serwlet, który wprawdzie niczego nie wyszukuje, ale przyjmuje dokument utworzony przez kod z listingu 13.12 i przesyła kryteria wyszukiwania z powrotem do przeglądarki. Do wydobycia ich z dokumentu wykorzystano obsługę XPath w JDOM.

Listing 13.13. Serwlet odbierający dokumenty XML

```
package javaxml3.ch13;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.xpath.XPath;

public class PostServlet extends HttpServlet {

    private SAXBuilder builder = new SAXBuilder();

    private XPath nameXPath;

    private XPath yearXPath;

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        if (!"text/xml".equals(request.getContentType())) {
            response.getWriter().println("Prześlij proszę jako text/xml.");
        } else {
            try {
                Document doc = builder.build(request.getReader());
```

```
                StringBuffer buff = new StringBuffer();
                buff.append("Szukano nazwy '" + nameXPath.valueOf(doc)
                        + "'");
                String year = yearXPath.valueOf(doc);
                if (!"notselected".equals(year)) {
                    buff.append(" i roku '" + year + "'");
                }
                buff.append(".");
                response.getWriter().print(buff.toString());
            } catch (JDOMException e) {
                response.getWriter().print(
                        "Błąd uzyskiwania kryteriów szukania: " + e.getMessage());
            }
        }
    }

    public void init() throws ServletException {
        try {
            nameXPath = XPath.newInstance("/search/name/text()");
            yearXPath = XPath.newInstance("/search/year/text()");
        } catch (JDOMException e) {
            throw new ServletException("Nie można utworzyć wyrażeń XPath.", e);
        }
        super.init();
    }
}
```

Wspólnie kod HTML i serwlet dadzą wynik jak na rysunku 13.10.



Rysunek 13.10. Wynik wysłania XML

Podrozdział ten zawiera krótkie wprowadzenie do XML w aplikacjach Ajax. Tworzenie programów przy użyciu tej techniki nie kończy się oczywiście na tym. Jest to bardzo dynamicznie rozwijająca się technologia, na której temat powstało mnóstwo zasobów dostępnych w internecie — od blogów (*Ajaxian.com*) po strony dla programistów prowadzone przez takie firmy jak Mozilla czy Microsoft. Dostępnych jest też na jej temat wiele publikacji książkowych, łącznie z, nie mógłbym nie wspomnieć, *Head Rush Ajax* autorstwa Bretta McLaughlina (O'Reilly).

Flash

Od chwili wprowadzenia w 1996 roku pod nazwą FutureSplash, Adobe Flash (wcześniej Macromedia Flash) stał się standardem dla twórców animacji i interaktywności sieciowej. Każde kolejne wydanie tej technologii było wzbogacane aż do stanu obecnego, w którym Flash jest w pełni funkcjonalną platformą do tworzenia dowolnego rodzaju animacji. Na platformę tę składają się obecnie trzy różne komponenty:

- format pliku o nazwie SWF (pliki w tym formacie są powszechnie nazywane filmami we Flashu),
- programy otwierające pliki SWF, które mogą występować jako samodzielne aplikacje (Flash Player) lub wtyczki do przeglądarek,
- narzędzia do tworzenia animacji.

Na stronie firmy Adobe dostępna jest specyfikacja formatu SWF. Jednak w warunkach licencji zabroniono wykorzystywania tej specyfikacji do tworzenia alternatywnego oprogramowania dla Flash Playera. Można jej używać przy pisaniu programów tworzących pliki SWF. W wyniku tego wśród narzędzi do tworzenia plików SWF można znaleźć produkty firmy Adobe (Flash Professional, Flash Standard i Flex) jak również kompilatory open source działające z poziomu wiersza poleceń, jak na przykład MTASC.

ActionScript

Prawie każda wersja Flasha zawierała obsługę języków skryptowych na jakimś poziomie. W 2005 roku we Flashu 5 wprowadzono nowy język skryptowy ActionScript.

ActionScript opiera się na ECMAScript, dzięki czemu przypomina JavaScript. We Flash 7 (znany również jako Flash MX 2004) wprowadzona została wersja 2.0 języka ActionScript, która posiadała właściwości bardziej znane z Javy niż z JavaScript, takie jak dziedziczenie klas, interfejsy i ścisła kontrola typów. ActionScript 3.0 pojawił się wraz z programem Flash Player 9 w 2006 roku i kontynuuje dzieło poprzednika poprzez ulepszenie mechanizmu obsługi wyjątków, dodanie rzeczywistego typowania podczas pracy, nowego API dla XML i obsługi wyrażeń regularnych.

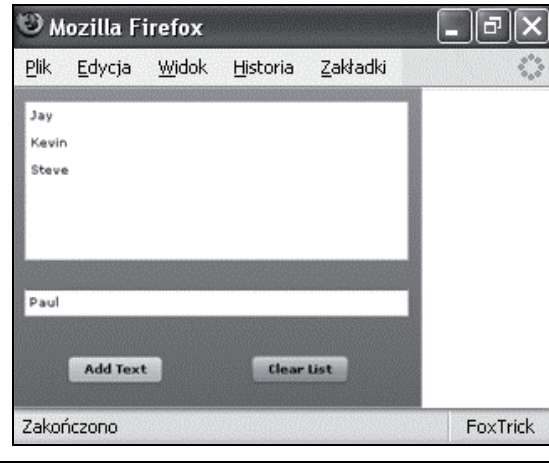
Ponadto wokół tego języka skupia się aktywna społeczność programistów. Można tworzyć dokumentacje w stylu Javadoc za pomocą narzędzia *as2api* i wykonywać testy jednostkowe za pomocą *AS2Unit*. Odnośniki do tych i innych narzędzi open source można znaleźć na stronie *http://www.osflash.org*.

Flex

W marcu 2004 roku firma Macromedia zaprezentowała oprogramowanie Flex, czyli alternatywny serwer mający stanowić platformę programistyczną Flasha. W przeciwieństwie do Flash Professional i Flash Standard, korzystających z binarnego formatu plików FLA, które są kompilowane do formatu SWF, Flex korzysta z formatu XML o nazwie MXML i z niego dokonuje kompilacji do formatu SWF. Pliki MXML reprezentują zarówno wygląd, jak i funkcjonalność aplikacji. Listing13.14 zawiera przykładowy plik MXML. Wynik kompilacji tego pliku widać na rysunku 13.11. Nawet osoby, które nie miały nigdy wcześniej do czynienia z językiem ActionScript, Flashem lub Fleksem, bez trudu odgadną, co robi ten program.

Listing 13.14. Prosty formularz w formacie MXML

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
    width="331" height="258">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var listData:ArrayCollection = new ArrayCollection();

            private function addText() : void {
                listData.addItem(text.text);
                text.text = "";
            }

            private function clearList() : void {
                listData.removeAll();
            }
        ]]>
    </mx:Script>

    <mx:List x="10" y="10" width="311" height="129" id="list"
        dataProvider="{listData}" />
    <mx:TextInput x="10" y="162" width="311" id="text"/>
    <mx:Button x="45" y="216" label="Dodaj tekst" click="addText()"/>
    <mx:Button x="194" y="216" label="Wyczyść listę" click="clearList()"/>
</mx:Application>
```



Rysunek 13.11. Prosty program MXML w przeglądarce

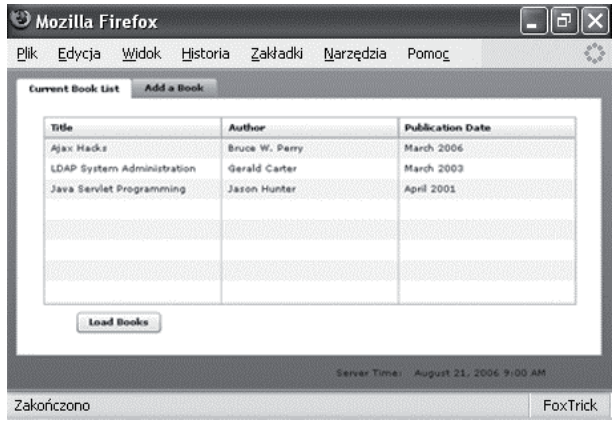
Pierwotnie Flex był dostępny wyłącznie jako aplikacja sieciowa J2EE. Kompilacja plików MXML do formatu SWF odbywała się w serwletach. Ponadto skompilowane aplikacje mogły używać bram dla danych, które były częścią tej aplikacji sieciowej, co dawało dostęp do takich zasobów serwerowych jak bazy danych, Enterprise JavaBeans (EJB) i oczywiście dane XML. Pierwsza wersja Fleksa była przeznaczona dla dużych firm i opłata licencyjna dla pojedynczej stacji roboczej była wysoka. Jednak z licencją serwerową pojawiło się narzędzie programistyczne o nazwie *Flex Builder*. Aczkolwiek, jako że MXML to tylko XML, a XML to tylko tekst, do pisania plików MXML wystarczy zwykły edytor tekstu.

W 2006 roku, z dniem wypuszczenia na rynek Fleksa 2.0, firma Adobe radykalnie zmieniła model i strukturę licencji Fleksa. Jego kompilator i dokumentację można bezpłatnie pobrać ze strony *http://www.adobe.com/products/flex* po uprzednim zarejestrowaniu się. Kompilatora można używać jako samodzielnej aplikacji do tworzenia plików SWF z plików MXML. Powstałe w ten sposób pliki SWF można umieszczać na dowolnych stronach (lub wcale ich nie używać na stronach). Nie jest wymagana żadna licencja serwerowa. Program Flex Builder 2 można zamówić jako odrębną aplikację i bazuje on na platformie Eclipse. Nadal jest licencjonowany na każdą stację komponent serwerowy dla Flex 2, który odpowiada za komunikację pomiędzy aplikacjami Flex a systemami zaplecza.

XML w ActionScript 3.0

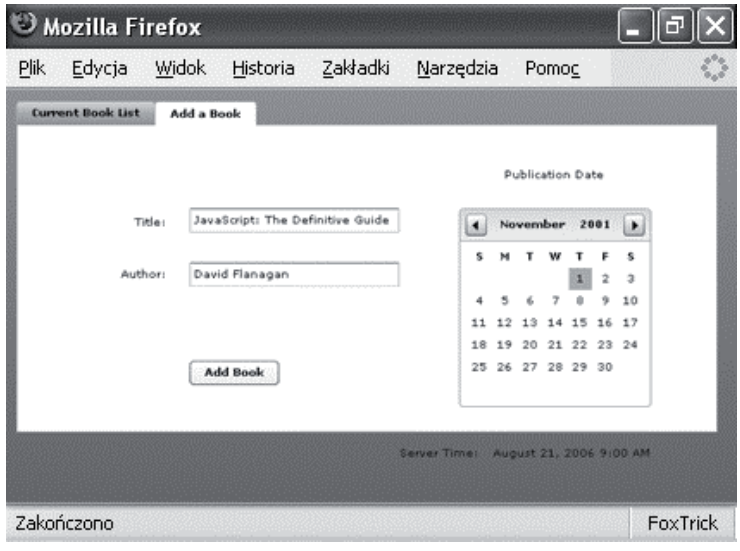
Pozostała część tego rozdziału opisuje kilka różnych sposobów pracy z dokumentami XML przy użyciu Flex 2 i ActionScript 3.0. Język ActionScript 3.0 jest obsługiwany przez Flash Player 9 i mające się ukazać środowisko Flash 9 oraz Flex 2 (w chwili pisania tej książki dostępna do pobrania na stronie *http://labs.adobe.com* była wersja alpha środowiska Flash Professional 9; zgodnie z informacjami zawartymi na tej stronie, jedynym powodem udostępnienia tej wczesnej wersji oprogramowania było umożliwienie tworzenia aplikacji przy użyciu ActionScript 3.0). Mimo że obecnie ActionScript 2.0 jest szeroko rozpowszechniony, niedługo sytuacja ta może ulec zmianie. Jako że format plików Fleksa bazuje na XML, do tematu tej książki i jej odbiorców bardziej pasuje Flex. Ponadto użycie oprogramowania Flex 2 oznacza, że przykłady będzie można skompilować za pomocą bezpłatnych narzędzi, co nie byłoby możliwe przy użyciu normalnego narzędzia Flash[2].

Aby zademonstrować omawiane właściwości stworzymy prostą aplikację prezentującą dane książek, wykorzystywane od początku tego rozdziału. Poza przeglądaniem listy książek będzie możliwość dodawania nowych. Na koniec, aby pochwalić się możliwościami użycia XML we Flashu, wyświetlone zostaną bieżąca data i godzina zgodnie z informacjami serwera. Rysunki 13.12 i 13.13 przedstawiają zrzuty ekranu zrobione podczas używania tego programu. Listing 13.15 zawiera kod MXML tworzący różne komponenty formularza. Podobnie jak w przypadku kodu Swing używanego w niektórych z poprzednich rozdziałów, nie trzeba go rozumieć również tutaj. W większości przypadków znaczenie jest oczywiste.



Rysunek 13.12. Strona z listą programu we Fleksie

[2] Mimo że dostępne są próbne wersje Flash Professional i Flash Standard, to nie jest to samo, co oprogramowanie bezpłatne.



Rysunek 13.13. Strona z formularzem we Fleksie

Listing 13.15. Komponenty formularza w MXML

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Script>
      <![CDATA[
        // Tutaj będzie znajdował się kod w języku ActionScript.
        ]]>
    </mx:Script>
    <mx:Canvas x="10" y="10" width="635" height="350">
        <mx:TabNavigator id="tabs" width="100%" height="300">
            <mx:Canvas label="Aktualna lista książek" width="100%" height="100%">
                <mx:Button x="65" y="221" id="load" label="Załaduj książki"/>
                <mx:DataGrid x="29" y="10" width="577" height="203" id="booklist">
                    <mx:columns>
                        <mx:DataGridColumn headerText="Tytuł"/>
                        <mx:DataGridColumn headerText="Autor"/>
                        <mx:DataGridColumn headerText="Data wydania"/>
                    </mx:columns>
                </mx:DataGrid>
            </mx:Canvas>
            <mx:Canvas label="Dodaj książkę" width="100%" height="100%"
              id="addFormPage">
                <mx:Label x="10" y="69" text="Tytuł:" width="128"
                  textAlign="right"/>
                <mx:TextInput x="156" y="65" width="190" id="title"/>
                <mx:Label x="10" y="116" text="Autor:" width="128"
                  textAlign="right"/>
                <mx:TextInput x="156" y="114" width="190" id="author"/>
                <mx:Label x="439.5" y="26" text="Data wydania"/>
                <mx:DateChooser x="400" y="65" id="pubdate"/>
                <mx:Button x="156" y="203" label="Dodaj" enabled="false"
                  id="add"/>
            </mx:Canvas>
        </mx:TabNavigator>
        <mx:Canvas y="300" width="100%" height="50">
```

```
            <mx:Label x="345" y="10" text="Czas serwera:"/>
            <mx:Text x="430" y="10" text="Czekaj..." width="191"
              textAlign="left"/>
        </mx:Canvas>
    </mx:Canvas>

</mx:Application>
```

E4X

ECMAScript for XML (E4X) to rozszerzenie standardu ECMAScript, na którym oparte są języki ActionScript i JavaScript. E4X dostarcza natywnej obsługi dokumentów XML w ECMAScript. Nowy dokument za pomocą E4X można utworzyć tak:

```
var doc=<content>
    <element attribute="1">
        <child>Tu jest trochę tekstu.</child>
    </element>
    <element attribute="2">
        <child>Tu powinno być jeszcze trochę tekstu.</child>
    </element>

</content>
```

W ActionScript lepiej jest dodać definicję typu, przez co pierwszy wiersz w kodzie powyżej powinien wyglądać tak:

```
var doc:XML=<content>
```

Dostęp do elementów potomnych uzyskuje się za pomocą notacji z kropką:

```
doc.element[0].child;
```

Dostęp do atrybutów można uzyskać przy użyciu znaku @:

```
doc.element[0].@attribute;
```

Dzięki takim możliwościom dostępu można odczytać wartości z dokumentu lub go zmodyfikować:

```
doc.element[1].child = "Zmiana tekstu elementu child. ";
```

Dodatkowo poza dostępem indeksowym można także używać wyrażeń XPath:

```
doc.element[@attribute="2"].child = "Zmiana tekstu elementu child kiedy attribute = 2.";
```

Analiza składniowa jest wykonywana za pomocą konstruktora XML:

```
var docString:String = "<content><element><child>tekst</child></element></content>";
var doc:XML = new XML(docString);
```

E4X w Firefoksie

Poza językiem ActionScript, E4X został zaimplementowany jeszcze w silniku JavaScript przeglądarki Mozilla Firefox. Funkcjonalność ta jest dostępna w Firefoksie od wersji 1.5. Jednak aby móc wykorzystać E4X w JavaScript, trzeba używać specjalnego atrybutu `type` znacznika `script`:

```
<script type="text/javascript; e4x=1">
```

W chwili pisania tej książki Microsoft nie ogłosił zamiaru dołączenia obsługi E4X w żadnej z przyszłych wersji Internet Explorera.

Więcej ogólnych informacji na temat E4X można znaleźć na stronie organizacji ECMA *http://www.ecma-international.org/publications/standards/Ecma-357.htm*.

Dostawcy danych XML

ActionScript i Flex umożliwiają łatwe dowiązywanie treści kontrolek do fragmentów dokumentów XML. Na powyższym listingu 13.15 znajduje się kontrolka `DataGrid` z trzema kolumnami, które odpowiadają elementom `child` w dokumencie XML. Dzięki temu można automatycznie zapełnić kontrolkę `DataGrid` treścią XML. W tym przypadku tworzony jest wiązalny obiekt listy, a właściwość `dataProvider` kontrolki `DataGrid` została ustawiona na nazwę tego obiektu listy:

```
// Te dwa wiersze są wewnątrz znaczników <mx:script>.
[Bindable]
private var gridData:ArrayCollection = new ArrayCollection();

<!-- To jest zmodyfikowana kontrolka <mx:DataGrid>. -->
<mx:DataGrid x="29" y="10" width="577" height="203" id="booklist"
  dataProvider="{gridData}">
  <mx:columns>
    <mx:DataGridColumn headerText="Tytuł" dataField="title"/>
    <mx:DataGridColumn headerText="Autor" dataField="author"/>
    <mx:DataGridColumn headerText="Data wydania" dataField="pubdate"/>
  </mx:columns>
</mx:DataGrid>
```

Należy zwrócić uwagę, że każda kolumna została poinformowana, z którego elementu ma pobrać treść.

Wysyłanie i ładowanie kodu XML

W ActionScript dostępna jest generyczna klasa `URLLoader`, której można używać do ładowania danych z podanego adresu URL. Jest też pokrewna klasa o nazwie `URLRequest`, która kapsułkuje wszystkie parametry żądania HTTP. Podobnie jak w przypadku wcześniej omawianego obiektu `XMLHttpRequest`, można za pomocą `URLRequest` i `URLLoader` odszukać dane XML, jak również wysłać je w ciele żądania. W przykładowej aplikacji klasy te zostały użyte dwa razy.

Wpierw, gdy użytkownik kliknie przycisk *Załaduj książki*, musi nastąpić żądanie adresu URL i dodanie każdej książki zdefiniowanej w pobranym XML do utworzonej powyżej listy. Jako że adres ten będzie używany więcej niż raz, będzie przechowywany w zmiennej:

```
private var serviceURL:String = "http://localhost:8080/ch13-flex/books";
```

Następnie tworzymy funkcję `loadBooks()`, która będzie wywoływana, gdy użytkownik kliknie przycisk *Załaduj książki*:

```
function loadBooks() : void {
  // Wyczyszczenie danych z siatki.
  gridData.removeAll();

  // Ładowanie danych.
  var loader:URLLoader = new URLLoader();
  loader.addEventListener("complete", getCompleteListener);
  loader.load(new URLRequest(serviceURL));
}

<!-- Zmodyfikowana kontrolka <mx:Button>. -->
<mx:Button x="65" y="221" id="load" label="Załaduj" click="loadBooks()"/>
```

Jak widać, `URLLoader` podobnie jak obiekt `XMLHttpRequest` w trybie asynchronicznym, wymaga powiadomienia procedury nasłuchującej zdarzeń o zakończeniu żądania. Podczas gdy `XMLHttpRequest` ma tylko jedną funkcję ustawioną jako jego `onreadystatechange`, `URLLoader` może mieć różne funkcje przypisane do każdego z sześciu zdarzeń, które definiuje. Lista tych zdarzeń znajduje się w tabeli 13.2. Każde zdarzenie ma nazwę i skojarzoną z nim klasę. Egzemplarz tej klasy jest przekazywany do zdefiniowanej funkcji. W przypadku zdarzenia `complete` klasa zdarzenia to generyczna klasa `flash.events.Event`. W związku z tym nasza funkcja `getCompleteListener()` wygląda tak:

```
function getCompleteListener(event:Event) : void {
  // Rzutowanie celu zdarzenia na obiekt URLLoader.
  var loader:URLLoader = URLLoader(event.target);
  parseBookDoc(new XML(loader.data));
}
```

Tabela 13.2. Zdarzenia klasy URLLoader

Nazwa zdarzenia	Klasa zdarzenia
`complete`	`flash.events.Event`
`httpStatus`	`flash.events.HTTPStatusEvent`
`ioError`	`flash.events.IOErrorEvent`
`open`	`flash.events.Event`
`progress`	`flash.events.ProgressEvent`
`securityError`	`flash.events.SecurityErrorEvent`

Oddzieliłem kod analizujący składnię dokumentu od funkcji nasłuchującej, dzięki czemu będzie można użyć jej ponownie później. W funkcji `parseBookDoc()` przechodzimy przez elementy `book` w dokumencie XML i dodajemy wszystkie do obiektu `gridData`.

```
function parseBookDoc(docXML:XML) : void {
  gridData.removeAll();
  for (var i:String in docXML.book) {
    gridData.addItem(docXML.book[i]);
  }
```

Drugie użycie klasy `URLLoader` związane jest z wysyłaniem informacji o nowej książce w postaci XML do serwera. Wpierw dokument XML jest składany za pomocą E4X. Jak powyżej, tworzymy funkcję, która zostanie wywołana po naciśnięciu przycisku *Dodaj*:

```
function addBook() : void {
  var newBookXML:XML = <book />
  newBookXML.title = title.text;
  newBookXML.author = author.text;

  var pickedDate:Date = pubdate.selectedDate;
  if (pickedDate == null)
    pickedDate = new Date();
  newBookXML.pubdate = pubdateFormatter.format(pickedDate);

  var request:URLRequest = new URLRequest(serviceURL);
  request.method="POST";
  request.contentType="text/xml";
  request.data = newBookXML.toXMLString();
  var loader:URLLoader = new URLLoader();
  loader.addEventListener("complete", postCompleteListener);
  loader.load(request);
}
```

```
function postCompleteListener(event:Event) : void {
  var loader:URLLoader = URLLoader(event.target);
  parseBookDoc(new XML(loader.data));
}

<!-- Zmodyfikowana kontrolka <mx:Button>. -->
<mx:Button x="156" y="203" label="Dodaj" enabled="false" id="add"
  click="addBook()"/>
```

Używanie gniazd XML we Flashu

Ostatni fragment kodu XML otwiera gniazdo do serwera i nasłuchuje dokumentów XML. W tym przypadku dokumenty te będą zawierały bieżącą datę i godzinę, na przykład:

```
<current>
    <date>25 sierpień 2006</date>
    <time>6:00:03 PM EDT</time>
</current>
```

Nasz program musi przy starcie otworzyć gniazdo i zaktualizować widok za każdym razem, gdy odbierze od serwera nowy dokument. Do tego celu można by było użyć klasy `URLLoader` za pomocą której można regularnie pytać serwer o nowy dokument XML. Jednak dzięki stałemu połączeniu gniazdowemu (ang. *socket connection*) serwer ma możliwość kontrolowania, kiedy i jak często wysyłane są dokumenty. Wysyłanie i odbieranie dokumentów XML poprzez stałe połączenie gniazdowe w ActionScript możliwe jest dzięki klasie o nazwie `XMLSocket`.

Zanim będzie można użyć klasy `XMLSocket`, trzeba wpierw napisać serwer gniazd (ang. *socket server*). Klasa z listingu 13.16 nie stanowi najlepszej implementacji serwera, ale na potrzeby tego przykładu wystarczy. Gdyby tego typu serwer miał być rzeczywiście używany, to trzeba by było pomyśleć o wielowątkowości. Więcej informacji na temat pisania serwerów gniazd można znaleźć w książce pod tytułem *Java Network Programming* autorstwa Eliotte Rusty Harold (O'Reilly).

Listing 13.16. Prosty serwer gniazd

```
package javaxml3;

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.net.BindException;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.DateFormat;
import java.util.Date;

import org.jdom.Document;
import org.jdom.Element;
import org.jdom.output.XMLOutputter;

public class DateTimeSocketServer extends Thread {

    private Document currentDocument;

    private Element dateElement;

    private DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG);

    private XMLOutputter outputter;
```

```
private int port;

private Element timeElement;

private DateFormat timeFormat = DateFormat.getTimeInstance(DateFormat.LONG);

public DateTimeSocketServer(int port) {
    this.port = port;
    outputter = new XMLOutputter();
    currentDocument = new Document();
    Element root = new Element("current");
    currentDocument.setRootElement(root);
    dateElement = new Element("date");
    root.addContent(dateElement);
    timeElement = new Element("time");
    root.addContent(timeElement);
}

public void run() {
    try {
        ServerSocket server = new ServerSocket(port);
        System.out.println("Nasłuchiwanie połączeń na porcie "
                + server.getLocalPort());

        while (true) {
            try {
                Socket socket = server.accept();

                BufferedWriter out = new BufferedWriter(
                        new OutputStreamWriter(socket.getOutputStream()));
                while (!socket.isClosed()) {
                    String current = updateCurrentXML();
                    System.out.println("Wysyłanie XML.");
                    out.write(current);
                    // Na końcu musi być bajt zerowy, aby Flash wiedział, że dokument
                    // już się skończył.
                    out.write(0);
                    out.flush();

                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                    }
                }
            } catch (IOException e) {
                System.err.println("Wystąpił wyjątek podczas przyjmowania: "
                        + e.getMessage());
            }
        }

    } catch (BindException e) {
        System.err.println("Nie można uruchomić serwera. Port jest zajęty.");
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }

}

private String updateCurrentXML() {
    Date date = new Date();
    dateElement.setText(dateFormat.format(date));
    timeElement.setText(timeFormat.format(date));
    return outputter.outputString(currentDocument);
}
```

```
    public static void main(String[] args) {
        int port;
        try {
            port = Integer.parseInt(args[0]);
        } catch (Exception ex) {
            port = 8900;
        }

        Thread t = new DateTimeSocketServer(port);
        t.start();
    }

}
```

Kluczowe znaczenie ma tutaj wysłanie bajta zerowego po dokumencie XML. Jeśli go zabraknie, to Flash nie będzie wiedział, że dokument już się skończył.

Aby połączyć się z serwerem, kiedy aplikacja jest załadowana, tworzymy metodę `init()` i robimy do niej odniesienie w znaczniku `mx:Application`:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
  initialize="init()">

private var xmlsocket:XMLSocket = new XMLSocket();

function init() : void {
  xmlsocket.addEventListener("connect", onXMLConnect);
  xmlsocket.addEventListener("data", onXMLData);
  xmlsocket.connect("localhost", 8900);
}
```

Podobnie jak w przypadku używania klasy `URLLoader`, trzeba utworzyć procedury nasłuchujące zdarzeń. `XMLSocket` i `URLLoader` mają wiele wspólnych zdarzeń. Te, które są dostępne w `XMLSocket`, wymieniono w tabeli 13.3.

Tabela 13.3. Zdarzenia klasy XMLSocket

Nazwa zdarzenia	Klasa zdarzenia
close	flash.events.Event
connect	flash.events.Event
data	flash.events.DataEvent
ioError	flash.events.IOErrorEvent
securityError	flash.events.SecurityErrorEvent

Pełny kod MXML tej aplikacji przedstawia listing 13.17.

Listing 13.17. Pełny kod MXML

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
  initialize="init()">
  <mx:Script><![CDATA[
            import mx.formatters.DateFormatter;
            import mx.collections.ArrayCollection;

            private var serviceURL:String =
              "http://localhost:8080/ch13-flex/books";

            [Bindable]
            private var gridData:ArrayCollection = new ArrayCollection();
```

```
private var pubdateFormatter:DateFormatter = new DateFormatter();

private var xmlsocket:XMLSocket = new XMLSocket();

function init() : void {
    pubdateFormatter.formatString = "MMMM YYYY";
    xmlsocket.addEventListener("connect", onXMLConnect);
    xmlsocket.addEventListener("data", onXMLData);
    xmlsocket.connect("localhost", 8900);
}

function onXMLConnect(event:Event) : void {
    serverTime.text = "Połączono z serwerem.";
}

function onXMLData(event:DataEvent) : void {
    var data:XML = XML(event.data);
    serverTime.text = data.date + " " + data.time;
}

function loadBooks() : void {
    // Wyczyszczenie danych z siatki.
    gridData.removeAll();

    // Ładowanie danych.
    var loader:URLLoader = new URLLoader();
    loader.addEventListener("complete", getCompleteListener);
    loader.load(new URLRequest(serviceURL));
}

// Funkcja ta zostanie wywołana po załadowaniu URL danych.
function getCompleteListener(event:Event) : void {
    // Rzutowanie celu zdarzenia na obiekt URLLoader.
    var loader:URLLoader = URLLoader(event.target);
    parseBookDoc(new XML(loader.data));
}

function parseBookDoc(docXML:XML) : void {
    gridData.removeAll();
    for (var i:String in docXML.book) {
        gridData.addItem(docXML.book[i]);
    }
}

function validateAddForm() : void {
    add.enabled = ((title.text.length > 0) &&
      (author.text.length > 0));
}

function addBook() : void {
    disableForm();

    var newBookXML:XML = <book />
    newBookXML.title = title.text;
    newBookXML.author = author.text;

    var pickedDate:Date = pubdate.selectedDate;
    if (pickedDate == null)
        pickedDate = new Date();
    newBookXML.pubdate = pubdateFormatter.format(pickedDate);

    var request:URLRequest = new URLRequest(serviceURL);
    request.method="POST";
    request.contentType="text/xml";
```

```
            request.data = newBookXML.toXMLString();
            var loader:URLLoader = new URLLoader();
            loader.addEventListener("complete", postCompleteListener);
            loader.load(request);
        }

        function postCompleteListener(event:Event) : void {
            var loader:URLLoader = URLLoader(event.target);
            parseBookDoc(new XML(loader.data));
            reenableForm();
        }

        function disableForm() : void {
            title.enabled = false;
            author.enabled = false;
            pubdate.enabled = false;
            add.enabled = false;
            load.enabled = false;
        }

        function reenableForm() : void {
            load.enabled = true;
            title.enabled = true;
            author.enabled = true;
            pubdate.enabled = true;
            title.text = "";
            author.text = "";
        }
    ]]></mx:Script>
  <mx:Canvas x="10" y="10" width="635" height="350">
    <mx:TabNavigator id="tabs" width="100%" height="300">
      <mx:Canvas label="Bieżąca lista książek" width="100%" height="100%">
        <mx:Button x="65" y="221" id="load" label="Załaduj książki"
          click="loadBooks()"/>
        <mx:DataGrid x="29" y="10" width="577" height="203" id="booklist"
          dataProvider="{gridData}">
          <mx:columns>
            <mx:DataGridColumn headerText="Tytuł" dataField="title"/>
            <mx:DataGridColumn headerText="Autor" dataField="author"/>
            <mx:DataGridColumn headerText="Data wydania" dataField="pubdate"/>
          </mx:columns>
        </mx:DataGrid>
      </mx:Canvas>
      <mx:Canvas label="Dodaj książkę" width="100%" height="100%" id="addFormPage">
        <mx:Label x="10" y="69" text="Tytuł:" width="128" textAlign="right"/>
        <mx:TextInput x="156" y="65" width="190" id="title"
          change="validateAddForm()"/>
        <mx:Label x="10" y="116" text="Autor:" width="128" textAlign="right"/>
        <mx:TextInput x="156" y="114" width="190" id="author"
          change="validateAddForm()"/>
        <mx:Button x="156" y="203" label="Dodaj" enabled="false" id="add"
          click="addBook()"/>
        <mx:Label x="439.5" y="26" text="Data wydania"/>
        <mx:DateChooser x="400" y="65" id="pubdate"/>
      </mx:Canvas>
    </mx:TabNavigator>
    <mx:Canvas y="300" width="100%" height="50">
      <mx:Label x="345" y="10" text="Server Time:"/>
      <mx:Text x="430" y="10" text="21 sierpień 2006 9:00 AM" width="191"
        textAlign="left" id="serverTime"/>
    </mx:Canvas>
  </mx:Canvas>
</mx:Application>
```

Dostęp z różnych domen

Domyślnie w programie Flash Player są dość ścisłe reguły definiujące, jakiego rodzaju dostęp do sieci jest dozwolony. Aplikacja Flash może tylko wysyłać żądania do hosta o **dokładnie** takiej samej nazwie jak ten, z którego została załadowana. Jeśli film we Flashu zostanie załadowany ze strony *http://www.przyklad.com/flash/movie.swf* to może wysyłać żądania do *http://www.example.com/feed* ale nie do *http://www.innastrona.com/feed*, ani nawet *http://data.przyklad.com/feed*. Aby otworzyć dostęp do innych hostów trzeba utworzyć plik ***cross-domain policy***. Gdyby źródło danych znajdowało się pod adresem *http://www.innastrona.com* i chcielibyśmy dać dostęp do filmów Flash załadowanych z *www.example.com*, to plik ten powinien być dostępny do pobrania pod adresem *http://www.innastrona.com/crossdomain.xml*:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
  "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
   <allow-access-from domain="www.przyklad.com" />
</cross-domain-policy>
```

Można stosować wiele elementów `allow-access-from-domain`. Ponadto można dzięki użyciu wyrażeń wieloznacznych dać dostęp do filmów Flash ładowanych z dowolnej poddomeny *przyklad.com*:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
  "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
   <allow-access-from domain="*.przyklad.com" />
</cross-domain-policy>
```

Używając wyrażeń wieloznacznych, można zezwolić na dostęp do filmów Flash z dowolnej domeny:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
  "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
   <allow-access-from domain="*" />
</cross-domain-policy>
```

Więcej na temat plików *cross-domain policy* można znaleźć na stronie *http://www.adobe.com/cfusion/knowledgebase/index.cfm?id=tn_14213*.

Na temat technologii omówionych w tym rozdziale można napisać znacznie więcej. Mam nadzieję, że to, co przedstawiłem, dało ogólne wyobrażenie na temat miejsca XML w warstwie prezentacyjnej aplikacji sieciowych oraz że podane przeze mnie źródła okażą się przydatne w poszukiwaniu dodatkowych informacji na poruszone tematy.